An Unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé, Tom Bouvier, Frédéric Mazoit, Nicolas Nisse, Ronan Pardo Soares

An Unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé, Tom Bouvier, Frédéric Mazoit, Nicolas Nisse, Ronan Soares

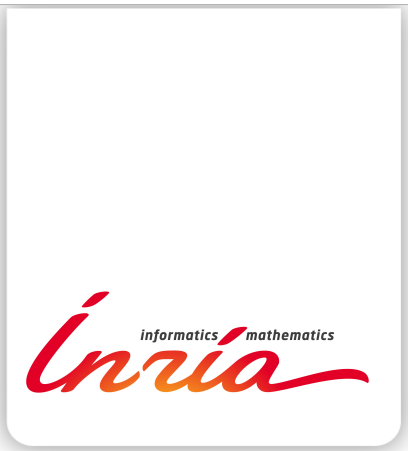
RESEARCH REPORT

N° 8372

September 2013

Project-Teams COATI

ISSN 0249-6399 ISRN INRIA/RR--8372--FR+ENG



An Unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé*, Tom Bouvier†, Frédéric Mazoit†, Nicolas Nisse‡§, Ronan Soares§‡¶

Project-Teams COATI

Research Report n° 8372 — September 2013 — 65 pages

Abstract: During the last decades, several polynomial-time algorithms have been designed that decide whether a graph has tree-width (resp., path-width, branch-width, etc.) at most k, where k is a fixed parameter. Amini *et al.* (Discrete Mathematics'09) use the notions of partitioning-trees and partition functions as a generalized view of classical decompositions of graphs, namely tree decomposition, path decomposition, branch decomposition, etc. In this paper, we propose a set of simple sufficient conditions on a partition function Φ, that ensures the existence of a linear-time explicit algorithm deciding if a set A has Φ-width at most k (k fixed). In particular, the algorithm we propose unifies the existing algorithms for tree-width, path-width, linear-width, branch-width, carving-width and cut-width. It also provides the first Fixed Parameter Tractable linear-time algorithm to decide if the q-branched tree-width, defined by Fomin *et al.* (Algorithmica'09), of a graph is at most k (k and q are fixed). Moreover, the algorithm is able to decide if the special tree-width, defined by Courcelle (FSTTCS'10), is at most k, in linear-time where k is a Fixed Parameter. Our decision algorithm can be turned into a constructive one by following the ideas of Bodlaender and Kloks (J. of Alg. 1996).

Key-words: Tree decomposition, FPT-algorithm, width-parameters, partitioning-tree, characteristics

* LIFO, ENSI-Bourges, Université d'Orléans, Bourges, France
† LaBRI, Université de Bordeaux, France
‡ Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
§ Inria, France
¶ ParGO, UFC, Brazil

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

An Unified FPT Algorithm for Width of Partition Functions

Résumé : Au cours de ces dernières années, plusieurs algorithmes polynomiaux ont été conçus pour décider si un graphe a largeur arborescente (resp., largeur en chemin, branch-width, etc) au plus k, où k est un paramètre fixe. Amini et al. (Discrete Mathematics'09) ont utilisé les notions d'arbres de partition et de fonctions de partition comme une vision généralisée des décompositions des graphes classiques, à savoir la décomposition arborescente, la décomposition en chemin, la décomposition en branche, etc. Dans cet article, nous proposons un ensemble de conditions sur une fonction de partition Φ, qui assure l'existence d'un algorithme explicite en temps linéaire pour décider si un ensemble A a Φ-largeur au plus k (oú k est fixé). En particulier, l'algorithme que nous proposons unifie les algorithmes existants pour la largeur arborescente, largeur en chemin, la largeur linéaire, la largeur de branche, cut-width et carving-width. Il est également le premier algorithme FPT pour décider si la largeur arborescente q-ramifié, définie par Fomin et al. (Algorithmica'09), d'un graphe est au plus k (k et q sont fixées). De plus, l'algorithme est capable de décider si la largeur arborescente spéciale, définie par Courcelle (FSTTCS'10), est plus k, où k est un paramètre fixé. Notre algorithme de décision peut être transformé en un algorithme constructif en suivant les idées de Bodlaender et Kloks (J. of Alg., 1996).

Mots-clés : Decomposition arborescente, algorithme FPT, largeur des graphes, arbres de partition, characteristiques

1 Introduction

Computing tree width. The notion of *tree width* is central in the theory of the Graph Minors developed by Robertson and Seymour [RS83,RS04]. Roughly, the tree-width of a graph measures how close a graph is to a tree. More formally, a *tree decomposition* $(T, \mathcal{X})$ of a graph $G = (V, E)$ is a tree T together with a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets of V, such that:

1. $\bigcup_{t \in V(T)} X_t = V$,
2. for any edge $e = \{u, v\} \in E$, there is $t \in V(T)$ such that $u, v \in X_t$, and
3. for any $v \in V$, the set of t such that $v \in X_t$ induces a subtree of T.

The width of $(T, \mathcal{X})$ is the maximum size of X_t minus 1, $t \in V(T)$, and the tree width $\mathrm{tw}(G)$ of a graph G is the minimum width among its tree decompositions. If T is restricted to be a path, we get a path decomposition of G, and the *path width* $\mathrm{pw}(G)$ of G is the minimum width among its path decompositions.

The notion of tree width plays an important role in the domain of algorithmic computational complexity. Indeed, many graph theoretical problems that are NP-complete in general are tractable when input graphs have bounded tree width. In this context, many *Fixed Parameter Tractable* (FPT) algorithms have been designed to solve problems like Hamiltonian Circuit, Independent Set, Graph Coloring, etc. More generally, the celebrated theorem of Courcelle states that any monadic second-order graph properties can be decided in linear time in the class of graphs of bounded tree width [CM93]. Typically, these algorithms are based on dynamic programming on a given tree decomposition of a graph, and use linear time in the number of vertices, but at least exponential in the width of the given decomposition of the input graph.

Thus, an important challenge consists in computing tree decompositions of graphs with small width. Heuristics and approximation algorithms have been designed (see, e.g., [BK07, FHL08]). Much research has been done on the problem of finding an optimal tree decomposition. This problem is NP-complete [ACP87] and special interest has been directed toward special graph classes [Bod96, BM93, BKK95]. The case of the class of graphs with bounded tree width has been widely studied in the literature [ACP87, Ree92].

In their seminal work on Graph Minors [RS83, RS04], Robertson and Seymour give a non-constructive proof of the existence of a $O(n^2)$ decision algorithm for the problems of deciding whether a graph belongs to some minor-closed class of graphs. Given that, for any k, the class of graphs of tree width at most k is minor-closed, an immediate consequence is the existence of a polynomial-time algorithm deciding whether a graph has tree width at most k, where k is a fixed parameter. However, such an algorithm is not given explicitly [RS86].

In [BK96], Bodlaender and Kloks design a linear time algorithm for solving this problem. More precisely, k and k' being fixed, given a n-node graph G and a tree decomposition of width at most k' of G, the Bodlaender and Kloks' algorithm decides if $\mathrm{tw}(G) \leq k$ in time $O(n)$. The big-oh hides a constant more than exponential in k and k'. In the last decades, analogous algorithms have been designed for other width parameters of graphs like path width [BK96], branch width [BT97], linear width [BT04], carving width and cut width [TSB00]. These algorithms are mainly based on the notion of *characteristic* (see Section 5).

Graph searching games. Both path width and tree width have also a nice theoretical-game interpretation. path width can be described as a graph searching game where a team of searchers aims at capturing an invisible and arbitrary fast fugitive hidden on the vertices of the graph, whereas tree width deals with the capture of a visible fugitive (see [Bie91, FT08] for surveys). In [FFN09], Fomin *et al.* introduce a variant of these games, called *non-deterministic graph searching*, that establishes a link between path width and tree width. Loosely speaking, in non-deterministic graph searching, the fugitive is invisible, but the searchers are allowed to query an oracle that possesses complete information about the position of the fugitive. However, the number of times the searchers can query the oracle is limited. The *q-limited search number* of a graph G, denoted by $\mathrm{s}_q(G)$, is the smallest number of searchers required to capture an invisible fugitive in G, performing at most $q \geq 0$ queries to the oracle. Fomin *et al.* give the following interpretation of non-deterministic graph searching in terms of graph decomposition. A tree decomposition $(T, \mathcal{X})$ is *q-branched* if T can be rooted in such a way that any path from the root to a leaf contains at most $q \geq 0$ vertices with at least two children (possibly, q may be unbounded in which case we set $q = \infty$).

Definition 1. Let G be a graph and $q \geq 0$ being fixed, the *q-branched tree width* $\mathrm{tw}_q(G)$ of a graph G is the minimum width among its q-branched tree decompositions.

By extension, the usual notion of tree width corresponds to the case $q = \infty$, i.e., $\mathrm{tw}_\infty(G) = \mathrm{tw}(G)$, while the path width corresponds to the case $q = 0$, i.e., $\mathrm{tw}_0(G) = \mathrm{pw}(G)$. For any $q \geq 0$ and any graph G, $\mathrm{s}_q(G) = \mathrm{tw}_q(G) + 1$ [FFN09, MN08]. Fomin *et al.* prove that deciding $\mathrm{s}_q(G)$ is NP-complete for any $q \geq 0$, and design an algorithm that decides whether $\mathrm{s}_q(G) \leq k$ in time $O(n^{k+1})$ for any n-node graph G [FFN09]. Prior to this work, no explicit FPT algorithm for this problem was known.

Partitioning trees. This paper aims at unifying and generalizing the FPT algorithms for computing various decompositions of graphs. As a particular application, our algorithm decides in linear time if the q-limited search number of a graph G is at most k, $q \geq 0$ and $k \geq 1$ fixed.

In order to generalize the algorithm of [BK96], we use the notions of partition function and partitioning tree defined in [AMNT09]. Given a finite set A, a partition function Φ for A is a function from the set of partitions of A into the integers. A partitioning tree of A is a tree T together with a one-to-one mapping between A and the leaves of T. The Φ-width of T is the maximum $\Phi(\mathcal{P})$, for any partition $\mathcal{P}$ of A defined by the internal vertices of T, and the Φ-width of A is the minimum Φ-width of its partitioning trees. Partition functions are a unified view for a large class of width parameters like tree width, path width, branch width, etc. In [AMNT09] is given a simple sufficient property that a partition function for A must satisfy to ensure that either A admits a partitioning tree of width at most $k \geq 1$, or there exists a *k-bramble* (a dual structure), unifying and generalizing the duality theorems in [RS83, RS91, ST93, FT03].

In this paper, we extend the definition of Φ-width to the one of q-branched Φ-width of a set A. Then, we use the framework of [BK96] applied to the notions of partition functions and partitioning tree in order to design a unified linear-time algorithm that decides if a finite set has q-branched Φ-width at most k. Again, $q \geq 0$ and $k \geq 1$ are fixed parameters.

1.1 Our results

We propose a simple set of sufficient properties and an algorithm such that, for any k and q fixed parameters, and any partition function Φ satisfying these properties, our algorithm decides in time $O(|A|)$ if a finite set A has q-branched Φ-width at most k (Theorem 4). Since tree width, path width, branch width, cut width, linear width, and carving width can be defined in terms of Φ-width for some particular partition functions Φ that satisfy our properties (Theorem 5), our algorithm unifies the works in [BK96, BT97, TSB00, BT04]. Our algorithm generalizes the previous algorithms since it is not restricted to width parameters of graphs but works as well for any partition function (not restricted to graphs) satisfying some simple properties. Moreover, we show how the special tree width [Cou10] of a graph can be defined by a partition function. This implies that our algorithm can also be used to decide, in linear time, if the special tree width of a graph is not bigger than a constant k. Finally, it provides the first explicit linear-time algorithm that decides if a graph G can be searched in a non-deterministic way by k searchers performing at most q queries, for any $k \geq 1$, $q \geq 0$ fixed. Our decision algorithm can be turned into a constructive one by following the ideas of Bodlaender and Kloks [BK96].

1.2 Organization of the paper

In Section 2, we formally define the notions of partition functions and partitioning trees. Then, we present several width parameters of graphs in terms of partition functions (most of these results have been proved in [AMNT09]). Section 3 is devoted to the formal statement of our results. In Section 4, we show a method to describe all partitioning trees of Φ-width not bigger than k of a set A. Section 5 is dedicated to show how partitioning trees can be represented in an efficient manner. Then, in Section 6, we describe an algorithm that follows the method in Section 4, but using the efficient representation of partitioning trees of Section 5, to decide if a set A has Φ-width at most k, Φ being a partition function and k a fixed integer. Then, in Section 7, we briefly discuss the results in this paper with some perspectives into future work.

2 Partition Functions and Partitioning Trees

In this section, we present the notions of partition function and partitioning tree of a set, as defined in [AMNT09].

Let A be a finite set. A partition of A is a set of non-empty pairwise disjoint subsets of A whose union equals A. Let $\mathrm{Part}(A)$ be the set of all partitions of A. Let $\mathcal{P} = (A_i)_{i \leq r}$ and $\mathcal{Q} = (B_i)_{i \leq p}$ be two partitions of A. For any subset $A' \subseteq A$, the *restriction* $\mathcal{P} \cap A'$ of $\mathcal{P}$ to A' is the partition $(A_i \cap A')_{i \leq r}$ of A', with its empty parts removed. $\mathcal{Q}$ is a *subdivision* of $\mathcal{P}$ if, for any $j \leq p$, there exists $i \leq r$ with $B_j \subseteq A_i$.

Definition 2. A *partition function* Φ_A *for* A is a function from $\mathrm{Part}(A)$ into the integers. A partitioning function Φ_A is *monotone* if, for any subdivision $\mathcal{Q}$ of a partition $\mathcal{P}$ of A, $\Phi_A(\mathcal{P}) \leq \Phi_A(\mathcal{Q})$.

For the purpose of generalization, we would like a partition function to be defined independently from the set on which it is applied. In particular, we

would like that a partition function, for some set A, to induce some partition functions for any subset of A.

From now on, $\mathcal{A}$ denotes a set of finite sets closed under taking subsets. In other words, $\forall X \in \mathcal{A}$ and $\forall Y \subseteq X$ we have $Y \in \mathcal{A}$.

Definition 3. A (monotone) *partition function* Φ *over* $\mathcal{A}$ is a function that associates a (monotone) partition function Φ_A for A to any $A \in \mathcal{A}$.

Most of the results of the chapter do not depend on the set A. When this is the case, we simplify the notation of a *partition function* Φ by omitting the subscript.

Definition 4. A partition function Φ over $\mathcal{A}$ is *closed under taking subsets* if Φ associates a partitioning function $\Phi_{A'}$ for any $A' \subseteq A \in \mathcal{A}$ and, for any partition $\mathcal{P}$ of A, $\Phi_{A'}(\mathcal{P} \cap A') \leq \Phi_A(\mathcal{P})$, where $\Phi(A) = \Phi_A$.

In what follows, we define partitioning trees.

Definition 5. A *partitioning tree* (T, σ) of a set A is a tree T together with a one-to-one mapping σ between the elements of A and the leaves of T.

If T is rooted in $r \in V(T)$, the partitioning tree is denoted by (T, r, σ). Any internal (i.e. non leaf) vertex $v \in V(T)$ corresponds to a partition $\mathcal{T}_v$ of A, defined by the sets of leaves of the connected components of $T \setminus v$. Figure 1 shows an example of a partitioning tree. Similarly, any edge $e \in E(T)$ defines a bi-partition $\mathcal{T}_e$ of A. The Φ_A *width* of (T, σ) is the maximum $\Phi_A(\mathcal{T})$ where $\mathcal{T}$ is the partition defined by an internal vertex of T or an edge of T.

Definition 6. Let Φ be a partition function over $\mathcal{A}$. The Φ-*width* of a set $A \in \mathcal{A}$ is the minimum Φ_A width of its partitioning trees.

A *branching node* of tree T rooted in $r \in V(T)$ is either r or a vertex of T with at least two children. A tree T is q-branched if there exists a root $r \in V(T)$ such that any path from r to a leaf contains at most $q \geq 0$ branching nodes. For instance, T is 0-branched if and only if T is a path.

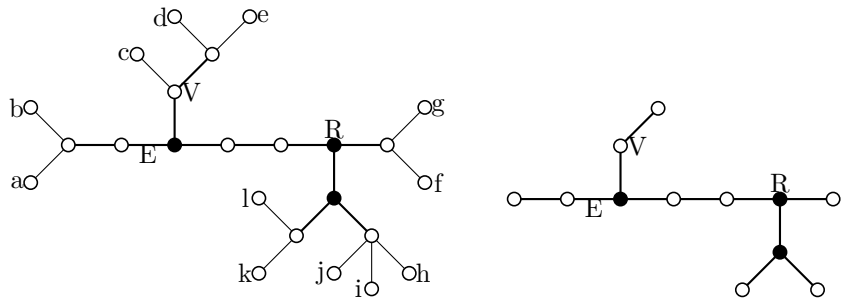
Definition 7. The *corpse* $\mathrm{cp}(T)$ of a tree T rooted in $r \in V(T)$ denotes the tree rooted in r obtained from T by removing all its leaves, but r if it is a leaf.

In Figure 1, a 2-branched partitioning tree (T, R, σ) of the elements $a, b, \ldots, k, l$ is represented. The vertex $V \in V(T)$ defines the partition $\mathcal{T}_V$ with parts $\{abfghijkl, c, de\}$, $R \in V(T)$ defines the partition $\mathcal{T}_R = \{abcde, fg, hijkl\}$, and the edge $E \in E(T)$ defines the bi-partition $\mathcal{T}_E = \{ab, cdefghijkl\}$. The black vertices are the branching nodes of $\mathrm{cp}(T)$.

Definition 8. A partitioning tree (T, σ) is q-*branched* if the corpse $\mathrm{cp}(T)$ of T is q-branched.

For instance, a partitioning tree (T, σ) is 0-branched if and only if T is a caterpillar[1]. The q-*branched* Φ-*width* of A is the minimum Φ_A width of its q-branched partitioning trees.

[1] A caterpillar is a tree with a dominating path.



(a) tree (b) corpse

Figure 1: A partitioning tree of $\{a, b, \ldots, k, l\}$ (a) and its corpse (b).

2.1 Graph Decompositions and Partitioning Trees

The notions from Section 2 have been given for general sets. In the following, we recall that partition functions and partitioning trees are generalization of several decompositions of graphs and their related parameters [AMNT09]. We assume that the reader is familiar the width measures of graphs such as tree width, branch width, cut width etc.

Throughout this section, $\mathcal{E}$ contains all possible edge-sets of every graph, i.e. for any graph $G = (V, E)$ we have $E \in \mathcal{E}$ and $\mathcal{V}$ contains all possible vertex-set of every graph, i.e. for any graph $G = (V, E)$ we have $V \in \mathcal{V}$.

It is sometimes necessary, depending on the width measure, to restrict the shape of the partitioning tree, to add some constraint to the mapping of the leaves of the partitioning tree or to use a special partitioning function in order to express graph width measures in terms of partitioning functions and partitioning trees. In what follows, we show which restrictions are necessary to represent the special tree width, branch width, linear width, cut width and carving width in terms of partitioning functions and partitioning trees. We start by reproducing how the q-branched tree width of a graph can be represented by partitioning functions as shown in [AMNT09].

2.1.1 Partition function and q-branched tree width

For any graph $G = (V, E)$, let Δ be the function that assigns, to any partition $\mathcal{X} = \{E_1, \ldots, E_r\}$ of E, the set of the vertices that are incident to edges in E_i and to edges in E_j, with $i \neq j$.

Definition 9. Let $E \in \mathcal{E}$. The function δ_E is the partition function for E that assigns $|\Delta(\mathcal{X})|$ to any partition $\mathcal{X}$ of $\mathrm{Part}(E)$. Let δ be the partition function over $\mathcal{E}$ that assigns δ_E to every $E \in \mathcal{E}$.

Lemma 1. *[AMNT09] For any graph $G = (V, E)$, the tree width $\mathrm{tw}(G)$ of G is at most $k \geq 1$ if, and only if, the δ width of E is at most $k + 1$.*

Proof. In other words, we aim at proving that for any graph $G = (V, E)$, the tree width $\mathrm{tw}(G)$ of G is at most k if, and only if, there is a partitioning tree of E with δ width at most $k + 1$. Let (T, σ) be a partitioning tree of E with δ width at most $k + 1$, then it is easy to check that $(\mathrm{cp}(T), (X_t)_{t \in V(\mathrm{cp}(T))})$, with $X_t = \Delta(\mathcal{T}_t)$, is a tree decomposition of G of width at most k. Conversely, let

$(T, \mathcal{X})$ be a tree decomposition of G with width at most k. Then, for any edge $\{x, y\} \in E$, let us choose an arbitrary bag X_t that contains both x and y, add a leaf f adjacent to t in T, and let $\sigma(f) = \{x, y\}$. Finally, let S be the minimal subtree spanning all such leaves. The resulting tree (S, σ) is a partitioning tree of E with δ width at most $k + 1$. □

A similar proof leads to:

Lemma 2. *[AMNT09] For any graph $G = (V, E)$, the path width $\mathrm{pw}(G)$ of G is at most $k \geq 1$ if, and only if, there is a 0-branched partitioning tree (T, σ) of E with δ width at most k.*

More generally:

Lemma 3. *For any graph $G = (V, E)$ and any $q \geq 0$, the q-branched tree width $\mathrm{tw}_q(G)$ of G is at most $k \geq 1$ if, and only if, there is a q-branched partitioning tree (T, σ) of E with δ width at most k.*

The special tree width can be represented with the following restriction over the partitioning trees.

Special tree width: The special tree width of a graph can be expressed in terms of the partition function δ, but with a restriction in the shape of the partitioning tree. For any graph $G = (V, E)$, instead of searching for the minimum δ_E width over all partitioning trees of E, we restrict the partitioning trees to respect the following rule. Let (T, σ) be a partitioning tree of E, for each vertex v in V, let T' be the minimum subtree of (T, σ) spanning all the leaves of (T, σ) such that their corresponding edge in E has v as one extremity. We have that T' is a caterpillar.

2.1.2 Other Widths and Partition Functions

The branch width and the linear width of a graph may be expressed in terms of the following partition function:

Definition 10. Let $max\delta_E$ be the partition function for $E \in \mathcal{E}$ which assigns $\max_{i \leq n} \delta(E_i, E \setminus E_i)$ to any partition $(E_1, \ldots, E_n)$ of E. Let $max\delta$ be the partition function over $\mathcal{E}$ that assigns $max\delta_E$ to any $E \in \mathcal{E}$.

Branch width [BT97]: By definition, the branch width of G, denoted by $\mathrm{bw}(G)$, is at most $k \geq 1$, if and only if there is a partitioning tree (T, σ) of E with $max\delta$ width at most k and such that the internal vertices of T have maximum degree at most three.

Linear width [BT04]: The linear width of G, denoted by $\mathrm{lw}(G)$ is defined as the smallest integer k such that E can be arranged in a linear ordering $(e_1, \ldots, e_m)$ such that for every $i = 1, \ldots, m-1$ there are at most k vertices both incident to an edge that belongs to $\{e_1, \ldots, e_i\}$ and to an edge in $\{e_{i+1}, \ldots, e_m\}$. The linear width of G is at most $k \geq 2$ if and only if there is a partitioning tree (T, σ) of E with $max\delta$ width at most k, such that the internal vertices of T have maximum degree at most three, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning tree of E and an ordering of E. Note that this does not hold for $k = 1$ (a 3 edges path is a counterexample).

The carving width of a graph may be expressed in terms of the following partition function. For any partition $\mathcal{X} = \{V_1, \ldots, V_r\}$ of $V \in \mathcal{V}$, let *Edgeδ* be the function that assigns the cardinality of the set of the edges of the graph $G = (V, E)$ that are incident to vertices in V_i and V_j, with $i \neq j$.

Definition 11. Let $maxEdge\delta_V$ be the partition function for $V \in \mathcal{V}$ that assigns $\max_{i \leq n} Edge\delta(V_i, V \setminus V_i)$ to any partition $(V_1, \ldots, V_n)$ of V. Let *maxEdgeδ* be the partition function over $\mathcal{V}$ that assigns $maxEdge\delta_V$ to any $V \in \mathcal{V}$.

Carving width [ST94, TSB00]: The carving width of G, carw(G), by definition, is at most $k \geq 1$ if and only if there is a partitioning tree of V with *maxEdgeδ* width at most k, and such that the internal vertices of T have maximum degree at most three.

The cut width of G, denoted by cw(G), is defined as the smallest integer k such that V can be arranged in a linear ordering $(v_1, \ldots, v_n)$ such that for every $i = 1, \ldots, n-1$ there are at most k edges both incident to a vertex that belongs to $\{v_1, \ldots, v_i\}$ and to a vertex in $\{v_{i+1}, \ldots, v_n\}$.

The partition function *maxEdgeδ* also may express the cut width of a graph $G = (V, E)$ when it is at least the maximum degree Δ of G. Note that any 0-branched partitioning tree with maximum degree at most 3 is such that its *maxEdgeδ*-width is at least Δ. On the other hand, any 0-branched partitioning with maximum degree at most 3 of V can be seen as a linear ordering over the vertices of G, which implies that the *maxEdgeδ*-width of G is at least as big as cw(G). More precisely, the minimum *maxEdgeδ* width of the 0-branched partitioning trees of V with maximum degree at most 3 equals $\max\{\text{cw}(G), \Delta\}$. In general, to express the cut width of a graph, we need a more restrictive partition function.

Definition 12. Let 3-$maxEdge\delta_V$ be the partition function for $V \in \mathcal{V}$ that assigns the function $\max\{Edge\delta(V_1, V \setminus V_1), Edge\delta(V_2, V \setminus V_2)\}$ to any partition (V_1, V_2, V_3) of V, with $|V_3| = 1$. Let 3-*maxEdgeδ* be the partition function over $\mathcal{V}$ that assigns 3-$maxEdge\delta_V$ to any $V \in \mathcal{V}$.

Cut width [TSB00]: The cut width of G is at most $k \geq 1$, if and only if there is a partitioning tree (T, σ) of V with 3-*maxEdgeδ* width at most k, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning tree of V and an ordering of V.

3 Main Results

In this section, we define properties of a partition function Φ that are sufficient for Φ to admit a linear-time (in the size of the input set) algorithm that decides whether the Φ-width of any set is at most k, k being a fixed integer.

More precisely, we start by giving a set of sufficient conditions for our theorem. Then, we show that all aforementioned widths respect these conditions. This implies that the algorithm in Section 6 can be used to compute all the aforementioned widths, thus this algorithm generalizes the FPT-algorithms of [BK96, BT97, TSB00, BT04].

3.1 Sufficient Conditions For a Linear Time Algorithm

First, some definitions are made in this section in order to state the main theorem.

Since partitioning trees generalize the tree decomposition to any set (not only graphs), it is natural to extend the notion of *nice tree decomposition [Bod96]* to any set.

A *nice decomposition* $(D, \mathcal{X})$ of a finite set A is a $O(|A|)$-node rooted tree D, together with a family $\mathcal{X} = (X_t)_{t \in V(D)}$ of subsets of A such that, $\cup_{t \in V(D)} X_t = A$, for all $a \in A$ the set $\{t \mid a \in X_t\}$ induces a subtree of D, and for any $v \in V(D)$:

start node: v is a leaf, or

introduce node: v has a unique child u, $X_u \subset X_v$ and $|X_v| = |X_u| + 1$, or

forget node: v has a unique child u, $X_v \subset X_u$ and $|X_u| = |X_v| + 1$, or

join node: v has exactly two children u and w, and $X_v = X_u = X_w$.

The *width* of a decomposition $(T, \mathcal{X})$ is the $\max_{t \in V(T)} |X_t|$. For any $v \in V(D)$, let D_v denote the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

Let Φ be a partition function. A *nice decomposition* $(D, \mathcal{X})$ of a set A is *compatible* with Φ if:

1. there exists a function F_Φ that associates an integer $F_\Phi(x, \mathcal{P}, e)$ to any integer x, partition $\mathcal{P}$ of some subset of A and element e of A, such that, F_Φ is strictly increasing in its first coordinate, and, for any introduce node $v \in V(D)$ with child u, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = F_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u).$$

2. there exists a function H_Φ that associates an integer $H_\Phi(x, y, \mathcal{P})$ to any pair of integers x, y, and partition $\mathcal{P}$ of some subset of A, such that, H_Φ is strictly increasing in its first and second coordinates, and, for any join node $v \in V(D)$ with children u and w, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = H_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \Phi_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

3. F_Φ and H_Φ have time complexity that does not depend on the size of A_v. That is, they have constant time complexity with respect to the size of A_v.

4. If $v \in V(D)$ is an introduction node with u as children, then for every partition $\mathcal{P}$ of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part, then $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_u}(\mathcal{P} \cap A_u)$.

   If $v \in V(D)$ is a join node with u and w as children, then for every partition $\mathcal{P}$ of A_v such that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part we have that $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_u}(\mathcal{P} \cap A_u)$. Respectively, if $\mathcal{P} \cap A_u$ is a partition of A_u with only one part, then $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_w}(\mathcal{P} \cap A_w)$.

Intuitively, the existence of F_Φ and H_Φ means that it is possible to quickly compute the Φ-width of some partitions $\mathcal{P}$ without knowing explicitly $\mathcal{P}$. By only knowing a restriction of $\mathcal{P}$ and the Φ-width of some restriction of $\mathcal{P}$, these restrictions being defined by the decomposition $(D, \mathcal{X})$. Moreover, the last restriction over the function Φ means that changes on the width of a partitioning tree resulted from adding elements to the partitioning tree do not propagate long. They are contained to vertices of the partitioning tree that partition X_v into at least two parts.

3.2 Main Theorem

This is the main theorem of this paper:

Theorem 4. *Let Φ be a monotone partition function that is closed under taking subsets. Let $k, k' \geq 1$ and $q \geq 0$ be three fixed integers (q may be ∞). There exists an algorithm that solves the following problem in time linear in the size of the input set:*

input: *a finite set A and a nice decomposition $(D, \mathcal{X})$, of width at most k' for A, that is compatible with Φ,*

output: *decide if the q-branched Φ-width of A is at most k.*

Corollary 1. *Let Φ be a monotone partition function that is closed under taking subsets. Let $k \geq 1$ and $q \geq 0$ be 2 fixed integers (q may be ∞). Let $\mathcal{A}$ be a class of sets such that there exists a linear-time algorithm for computing a nice decomposition of width $O(k)$ for any set $A \in \mathcal{A}$, compatible with Φ, if it exists. There exists an algorithm that solves the following problem in time linear in the size of the input set:*

input: *a finite set A,*

output: *decide if the q-branched Φ-width of A is at most k.*

The proof of Theorem 4 is quite technical and most of the remaining part of this paper is devoted to it. In order to explain such proof in a more didactic manner, we start with a simple algorithm to solve this problem, albeit not in linear time, and improve such algorithm in the following sections until we have a linear time algorithm.

3.3 Tractability of Width Parameters of Graphs

This section is devoted to present an application of Theorem 4 in terms of the width measures of graphs showed in Section 2.

Theorem 5. *Let k and q be two fixed parameters. There exists an algorithm that solves the following problem in time linear in the size of the input graph.*

input: *A graph G with maximum degree bounded by a function of q and k,*

output: *Decide if G has q-branched tree width, resp., branch width, linear width, carving width, cut width or special tree width at most k.*

Proof. In Section 2, we explained that several width parameters of graphs (q-branched tree width, resp., branch width, linear width, carving width, cut width or special tree width) can be defined in terms of partition functions. Therefore, the proof of Theorem 5 roughly consists in proving that the partition functions corresponding to these width parameters satisfy conditions of Theorem 4.

Bodlaender designs a linear-time algorithm that decides if the tree width of a graph G is at most k' (k' is a fixed parameter), and, if $\text{tw}(G) \leq k'$ returns a tree decomposition of width at most k' [Bod96]. Moreover, a nice tree decomposition of G can be computed in linear time from any tree decomposition of G, and without increasing its width [BK96]. Moreover, from Lemma 7, any nice tree decomposition of G can be turned into a nice decomposition of $E(G)$ with width at most dk', where d is the maximum degree of G.

Note that from Lemmas 8, 9 and 10 we have that a nice decomposition is compatible with partitioning functions for q-branched tree width, branch width, linear width, carving width, cut width and special tree width. Therefore, in order to obtain a nice decomposition of $E(G)$, we can use the algorithm in [Bod96] and Lemma 7. Since, by hypothesis, the maximum degree of G is bounded by a function of q and k, the width of the nice decomposition obtained is bounded by a fucntion of q and k. Therefore, the hypothesis of Theorem 4 is satisfied for all the aforementioned widths, which proves Theorem 5.

□

First, the following lemma is straightforward and its proof is thus omitted.

Lemma 6. *The partition functions δ, maxδ, Edgeδ and maxEdgeδ are monotone and closed under taking subsets.*

Next three lemmas show the compatibility of some nice decomposition with the partition functions δ, *max*δ and *maxEdge*δ.

Definition 13. [BK96] A nice tree decomposition of a graph $G = (V, E)$ is a tree decomposition of G that is a nice decomposition of V.

Lemma 7. *Any nice tree decomposition $(T, \mathcal{Y})$ of a graph $G = (V, E)$ with width k can be turned into a nice decomposition $(D, \mathcal{X})$ of E. Moreover, if G has bounded maximum degree d, the width of $(D, \mathcal{X})$ is at most $d \cdot k$.*

Proof. For any $v \in V(T)$, let T_v denote the subtree of T rooted in v, and $A_v = \cup_{t \in V(T_v)} Y_t$, and let E_v be the set of edges belonging to the subgraph induced by the vertices contained in A_v that are incident to a vertex in Y_v. Any start node, resp., join node, Y_t of $(T, \mathcal{Y})$ corresponds to a start node, resp., join node, E_t of $(D, \mathcal{X})$. For any introduce node Y_t of $(T, \mathcal{Y})$, let $x \in V$ be the vertex such that $Y_t = Y_{t'} \cup \{x\}$, where t' is the single child of t in T. Let $e_1, \ldots, e_r$ be the edges that are incident to x and to some vertex in $Y_{t'}$. Then, Y_t is modified into a path of introduce nodes $E(G[Y_{t'}]) \cup \{e_1\}, E(G[Y_{t'}]) \cup \{e_1, e_2\}, \ldots, E(G[Y_{t'}]) \cup \{e_1, e_2, \ldots, e_r\}$ in $(D, \mathcal{X})$. Finally, any forget node Y_t of $(T, \mathcal{Y})$ is modified into a path of forget nodes $E(G[Y_{t'}]) \setminus \{e_1\}, E(G[Y_{t'}]) \setminus \{e_1, e_2\}, \ldots, E(G[Y_{t'}]) \setminus \{e_1, e_2, \ldots, e_r\}$ in $(D, \mathcal{X})$, where t' is the unique child of t in T, and $e_1, \ldots, e_r$ are the edges that are incident to $x = Y_{t'} \setminus Y_t$ and to no other vertex in Y_t. The obtained decomposition of E is a nice decomposition and its width (i.e. the maximum number of edges in each bag) is at most the width of the tree decomposition $(T, \mathcal{Y})$ times the maximum degree of G. □

Lemma 8. *Let G be a graph with maximum degree deg. Given a nice tree decomposition $(T, \mathcal{Y})$ of G with width at most $k' \geq 1$, a nice decomposition $(D, \mathcal{X})$ of E, compatible with the partition functions corresponding to tree width (resp., path width and special tree width) and with $\max_{t \in V(D)} |X_t| \leq k' \cdot deg$ can be computed in linear time.*

Proof. Recall that the tree width, the path width and the special tree width of a graph may be defined in terms of the partition function δ. First, let $(D, \mathcal{X})$ be the nice decomposition of E, with width at most $k' \cdot deg$, obtained from $(T, \mathcal{Y})$ as indicated in Lemma 7. We aim at proving that $(D, \mathcal{X})$ is compatible with δ. Let F_δ be defined as follows.

Definition 14. Let x be an integer, $\mathcal{P}$ be a partition of a subset E' of E and an edge $e \in E'$. Let $F_\delta(x, \mathcal{P}, e) = x + |\{v \in e \mid v \in \Delta(\mathcal{P}) \setminus \Delta(\mathcal{P} \cap (E' \setminus \{e\}))\}|$.

That is, F_δ adds to x the number of vertices incident to e that contribute to the border of the partition $\mathcal{P}$ because they are incident to e. F_δ is obviously strictly increasing in its first coordinate. Moreover, F can be computed in constant time when $|E'|$ is bounded by a constant.

For any $v \in V(D)$, let D_v denote the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

Let $v \in V(D)$ be an introduce node with child u, and let $\{e\} = X_v \setminus X_u$. Let $\mathcal{P}$ be a partition of A_v. We need to prove that $\delta_{A_v}(\mathcal{P}) = F_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, e)$. In other words, let us prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta((\mathcal{P} \cap X_v) \cap (X_v \setminus \{e\}))\}|$.

$\delta_{A_v}(\mathcal{P})$ is the number of vertices in the subgraph induced by the set of edges A_v that are incident to edges in different parts of $\mathcal{P}$. This set of vertices can be divided into two disjoint sets: (1) the set S_1 of vertices that are incident to two edges f and h that are different from e and that belong to different parts of $\mathcal{P}$, and (2) the set S_2 of vertices x incident to e and such that all other edges (different from e) incident to x belong to the same part of $\mathcal{P}$ that is not the part of e. S_1 is exactly the set of vertices belonging to $\Delta_{A_u}(\mathcal{P} \cap A_u)$, therefore $|S_1| = \delta_{A_u}(\mathcal{P} \cap A_u)$.

By definition of $(D, \mathcal{X})$, because it has been built from a tree decomposition, any edge of $A_u = A_v \setminus \{e\}$ that has a common end with e belongs to X_v. Therefore, any vertex in S_2 belongs to $\Delta(\mathcal{P} \cap X_v)$. It is easy to conclude that $|S_2| = |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta(\mathcal{P} \cap X_v \cap (X_v \setminus \{e\}))\}|$.

Therefore, the function F_δ satisfies the desired properties.

Definition 15. Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset E' of E. Let $H_\delta(x, y, \mathcal{P}) = x + y - \delta(\mathcal{P})$.

H_δ is obviously strictly increasing in its first and second coordinates. Moreover, it can be computed in constant time when $|E'|$ is bounded by a constant.

Let $v \in V(D)$ be a join node with children u and w, and let $\mathcal{P}$ be a partition of A_v, we must prove that $\delta_{A_v}(\mathcal{P}) = H_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \delta_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v)$. That is, we prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + \delta_{A_w}(\mathcal{P} \cap A_w) - \delta_{X_v}(\mathcal{P} \cap X_v)$.

First, note that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w) \subseteq \Delta_{A_v}(\mathcal{P})$. Moreover, by definition of the nice decomposition $(D, \mathcal{X})$, an edge of $A_u \setminus X_v$ and an edge of $A_w \setminus X_v$ cannot be incident. Indeed, X_v has been built by taking all edges incident to a vertex in a bag Y of the tree decomposition $(T, \mathcal{Y})$. By the connectivity property of a tree decomposition, if a vertex x would have been incident to

an edge in $A_u \setminus A_w$ and to an edge in $A_w \setminus A_u$, then $x \in Y$ which would have implied that both these edges belong to $X_v = A_u \cap A_w$, a contradiction. Therefore, $\Delta_{A_v}(\mathcal{P}) \subseteq \Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w)$. To conclude showing that H_δ is correct, it is sufficient to observe that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cap \Delta_{A_w}(\mathcal{P} \cap A_w) = \Delta_{X_v}(\mathcal{P} \cap X_v)$.

Now, we need to show that if v is an introduction node with u as children then for all partitions $\mathcal{P}$ of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part then $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$.

Assume that v is an introduction node and that $A_v \setminus A_u = \{a\}$. Let $\mathcal{P}$ be a partitioning of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part.

Since $(D, \mathcal{X})$ is a nice decomposition of E, we have that a is not adjacent to any edge in $A_u \setminus X_v$. Assume that $\Delta_{A_u}(\mathcal{P} \cap A_u) < \Delta_{A_v}(\mathcal{P})$. This means that there is an extremity of a that is incident to an edge of $A_u \setminus X_v$, a contradiction. Therefore, $\Delta_{A_u}(\mathcal{P} \cap A_u) = \Delta_{A_v}(\mathcal{P})$.

Finally, we need to show that if v is a join node of $(D, \mathcal{X})$ with children u and w, then for all partitions $\mathcal{P}$ of A_v such that $\mathcal{P} \cap A_u$ is a partition of A_u with only one part then $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_w}(\mathcal{P} \cap A_w)$ or $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$ in the case that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part.

W.l.o.g. assume that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part. Since, $(D, \mathcal{X})$ is a nice decomposition of E, we have that no edges in $A_u \setminus A_w$ share extremities with edges in $A_w \setminus A_u$. Moreover, $(A_u \setminus A_w) \cap (A_w \setminus A_u) = \emptyset$ and $A_u \cap A_w = X_v$.

$\Delta_{A_v}(\mathcal{P})$ is the number of vertices of G such that they are extremities for edges in different parts of $\mathcal{P}$. Since, $\mathcal{P} \cap A_w$ is a partition with only one part and the edges of $A_w \setminus A_u$ do not share any extremities with edges in $A_u \setminus A_w$. We have that $\Delta_{A_v}(\mathcal{P})$ is the number of vertices that are extremities of edges in different parts of $\mathcal{P} \cap A_u$. That is, $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$.

The case where $\mathcal{P} \cap A_u$ is a partition of A_u with only one part is similar and thus omitted. □

It is easy to see that the partition function *Edgeδ* behaves as the δ function but the role of vertices and edges being reversed.

First note that any nice tree decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V. To prove that $(T, \mathcal{Y})$ is compatible with the partition function *Edgeδ*, we follow the above proof of Lemma 8.

Definition 16. $F_{Edge\delta}$ and $H_{Edge\delta}$ are defined as follows:

- Let x be an integer, $\mathcal{P}$ be a partition of a subset V' of V and a vertex $v \in V'$. Then,

  $$F_{Edge\delta}(x, \mathcal{P}, v) = x + |\{e \in E \mid v \in e \text{ and } e \in Edge\delta(\mathcal{P}) \setminus Edge\delta(\mathcal{P} \cap (V' \setminus \{v\}))\}|.$$

- Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset V' of V. Then, $H_{Edge\delta}(x, y, \mathcal{P}) = x + y - Edge\delta(\mathcal{P})$.

Therefore, the partition function *Edgeδ* is compatible with any nice tree decomposition. To prove the compatibility of the partition function *maxEdgeδ* with any nice tree decomposition, we use the following claim.

Let f be any partition function and let *maxf* be the partition function that associates $\max_{i \leq n} f(A_i, A \setminus A_i)$ to any partition $(A_1, \ldots, A_n)$ of A.

Claim 1. *For any partition function f compatible with a nice decomposition of some set A, the partition function $maxf$ is also compatible.*

Remark 1. Claim 1 also serves to show that the partition functions for branch width and linear width are also compatible to a nice decomposition of the edges of a graph.

Because f is compatible with any nice decomposition of A, there exist two function F_f and H_f that satisfy the properties defining the notion of compatibility. We must have:

$$f_{A_v}(\mathcal{P}) = F_f(f_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u), \text{ and}$$
$$f_{A_v}(\mathcal{P}) = H_f(f_{A_u}(\mathcal{P} \cap A_u), f_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

The key point is that if $\mathcal{P}$ is a bipartition of some set A, then $f_A(\mathcal{P}) = maxf_A(\mathcal{P})$. Therefore, when considering a bipartition, the functions F_{maxf} and H_{maxf} can be defined similarly to F_f and H_f. The case that $\mathcal{P}$ is not a bipartition is more technical, hence we postpone the proof of this claim until Section 6 after showing how the algorithm works.

Hence with Claim 1 the partition functions corresponding to the branch width and linear width (carving width and cut width) are all compatible to nice decompositions of E (V) of a graph $G = (V, E)$. This gives us the following lemmas:

Lemma 9. *Any nice decomposition $(D, \mathcal{X})$ of G is a nice decomposition of E compatible with the partition functions corresponding to branch width (resp., linear width).*

Lemma 10. *Any nice tree decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V compatible with the partition functions corresponding to carving width (resp., cut width).*

4 Describing Partitioning Trees in a Dynamic Manner

In this section, we show the basic idea used to compute all the aforementioned widths.

4.1 Preliminary Definitions

Definition 17. Let (T, r, σ) be a rooted partitioning tree of a set A and $A' \subseteq A$, then the *partitioning tree* (T', r', σ') *of* (T, r, σ) *restricted to* A' is the minimum subtree of T' spanning all the leaves corresponding to elements of A'. r' is the vertex of T' that is closest to r in T and the function σ' is the restriction of σ to A'.

Let $(D, \mathcal{X})$ be a nice decomposition of a set A and Φ a partitioning function of A that is compatible with $(D, \mathcal{X})$. Recall that for any $v \in V(D)$, D_v denotes the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$. In what follows, a *full set of partitioning trees* of a vertex $v \in V(D)$, denoted by $\mathrm{FSPT}_{\mathrm{k,q}}(v)$, is the set of

all labeled q-branched partitioning trees of Φ-width at most k of A_v. If r is the root of $(D, \mathcal{X})$ then $A_r = A$, hence $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ is not empty if and only if the q-branched Φ-width of A is not bigger than k.

Definition 18. A *labeled partitioning tree*, $((T, r, \sigma), \ell)$ is a partitioning tree (T, r, σ) along with a label ℓ, a function from the edges or internal vertices of T to integers, the label of a vertex (or edge) t of T is such that $\ell(t) = \Phi(\mathcal{A}_t)$, where $\mathcal{A}_t$ is the partition of A defined by t.

The role of the label is to store the values of the partitioning function Φ for fast access and update during the execution of the algorithm.

4.2 Main Idea

Let k and q be fixed integers. The main idea behind the algorithm in section Section 6 is to use dynamic programming to compute a full set of partitioning trees for A by using a nice decomposition $(D, \mathcal{X})$ of A. In other words, to decide if the q-branched Φ-width of A is not bigger than k. We start by computing a $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for all bags X_v where v is a leaf of D. Then, for each vertex $v \in V(D)$ such that for each child u of v we have already computed the set $\mathrm{FSPT}_{\mathrm{k,q}}(u)$, we compute $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Once $\mathrm{FSPT}_{\mathrm{k,q}}(r)$, where $r \in V(D)$ is the root of D, is computed, then we can simply test if $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ is empty to decide whether the q-branched Φ-width of A is not bigger than k.

In this section we show how to compute $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for vertices of $V(D)$, for the moment, we do not focus on the complexity of this computation, rather we show the main idea behind each procedure introduced on Section 6. The computation of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ depends on the type of the node v. In the following subsections we show procedures for each kind of node in the nice decomposition $(D, \mathcal{X})$ (starting node, introduce node, forget node and join node).

We also state that it is possible that a full set of characteristics has infinite size, hence it is necessary to design a method to "compress" this set reducing its cardinality to something more manageable, i.e., a size given by a function bounded on k, the width of the nice decomposition and q. In Section 5 we show how this can be achieved.

Then, in Section 6, we show how to use this compression to design a linear time algorithm to decide if the Φ-width of a set A is not bigger than k, k being a fixed parameter.

4.3 Procedure Starting Node

If v is a starting node, i.e. a leaf of D. Then, procedure Starting Node consists of enumerating all q-branched partitioning trees for A_v with Φ-width at most k.

That is, $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is the set of all possible labeled q-branched partitioning trees for A_v with Φ-width at most k.

Lemma 11. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A. The procedure Starting Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k for a starting node v of D.*

4.4 Procedure Introduce Node

Assume that v is an introduce node of D and that we aim at computing $\mathrm{FSPT}_{\mathrm{k,q}}(v)$.

Let u be the only child of v in D and $\{a\} = X_v \setminus X_u$. Let $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be the full set of labeled q-branched partitioning trees of A_u with width at most k. The set $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is obtained from $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ by applying the following procedure to every labeled partitioning tree (T_u, r_u, σ_u) in $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ and to every possible execution of the step "*update T_u into T_v*".

Procedure Introduce Node. Starting with $\mathrm{FSPT}_{\mathrm{k,q}}(v) =$, for all possible choices of step "update T_u into T_v" and for all elements$(T_u, r_u, \sigma_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ do the following:

update T_u into T_v: To insert a corresponding vertex to a in $((T_u, r_u, \sigma_u), \ell_u)$, choose some internal vertex v_{att} of $V(T_u)$, add a leaf v_{leaf} adjacent to v_{att}. Moreover, let $e_{new} = \{v_{att}, v_{leaf}\}$, we then proceed to subdivide e_{new} a finite number of times. Then, set $\sigma_v(v_{leaf}) = a$. Let P_{new} be the path joining v_{leaf} to v_{att}. If $r_u = v_{att}$ then r_v is one of the vertices in $V(P_{new}) \setminus \{v_{leaf}\}$, otherwise $r_v = r_u$. Note that, at this point, T_v is a partitioning tree of A_v.

update of labels of new vertex(s) and edge(s): First, let P_{new} be the path joining v_{leaf} to v_{att}. Then every internal vertex (or edge) p of P_{new} receives label $\ell_v(p) = \Phi_{A_v}(\{A_u, \{a\}\})$.

update of labels of other vertex(s) and edge(s): For all $e \in E(T_v) \setminus E(P_{new})$, let $\mathcal{T}_e$ be the partition of X_v defined by e. $\ell_v(e) \leftarrow F_\Phi(\ell_u(e), \mathcal{T}_e, a)$. For all $t \in (V(T_v) \setminus V(P_{new})) \cup \{v_{att}\}$, let $\mathcal{T}_t$ be the partition of X_v defined by t, then $\ell_v(t) \leftarrow F_\Phi(\ell_u(t), \mathcal{T}_t, a)$.

update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$: If $((T_v, r_v, \sigma_v), ell_v)$ is q-branched, for every internal vertex $t \in V(T_v)$ we have $\ell_v(t) \leq k$ and for every edge $e \in E(T_v)$ we have $\ell_v(e) \leq k$ then $\mathrm{FSPT}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSPT}_{\mathrm{k,q}}(v) \cup \{((T_v, r_v, \sigma_v), \ell_v)\}$, otherwise $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ remains unchanged.

Lemma 12. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be an introduce node of D with a child u. The procedure Introduce Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k from the set $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ for the node v.*

Proof. Let $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ be the set computed by the procedure Introduce Node, we first show that any element $((T_v, r_v, \sigma_v), \ell_v) \in \mathrm{FSPT}_{\mathrm{k,q}}(v)$ is a q-branched partitioning tree with Φ-width not bigger than k for A_v.

Let $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be a full set of q-branched partitioning trees of Φ-width not bigger than k for the node u and let (T_u, r_u, σ_u) be any element of $\mathrm{FSPT}_{\mathrm{k,q}}(u)$. Assume that $((T_v, r_v, \sigma_v), \ell_v)$ is obtained through an execution of procedure Introduce Node on (T_u, r_u, σ_u).

From the step "update T_u into T_v", since (T_u, r_u, σ_u) is a partitioning tree for A_u and $A_v \setminus A_u = \{a\}$, taking (T_u, r_u, σ_u) adding a leaf v_{leaf} to an internal vertex v_{att} of T_u and mapping v_{leaf} to a results in a partitioning tree for A_v. Moreover, the subdivision of $\{v_{att}, v_{leaf}\}$ does not change the fact that (T_v, r_v, σ_v) is a partitioning tree for A_v. Hence, (T_v, r_v, σ_v) is a partitioning tree for A_v.

For any internal vertex (or edge) t of T_v let $\mathcal{A}_t$ be the partition of A_v that it defines. It remains to show that (T_v, r_v, σ_v) is q-branched, has Φ-width not bigger than k and that after the execution of the procedure Introduce Node all labels are correct. In other words, for all internal vertices (or edges) t of T_v, we prove that $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$.

In the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is q-branched. Then, from the fact that there are only labeled partitioning trees in $\mathrm{FSPT}_{\mathrm{k,q}}(u)$, we have that $\ell_u(t) = \Phi_{A_u}(\mathcal{A}_t \cap A_u)$ for any internal vertex (or edge) t of T_u. Moreover, we have that Φ is compatible with the nice decomposition $(D, \mathcal{X})$. Therefore, from the description of the Introduce Node procedure for every internal node t (or any edge) of T_v that is not in $P_{new} \cup \{v_{att}\}$:

$$\ell_v(t) = F_\Phi(\ell_u(t), \mathcal{T}_t, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \mathcal{A}_t \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_t).$$

Furthermore, all edges and internal vertices of P_{new} receive the label $\Phi_{A_v}(\{A_u, \{a\}\})$ from step "update of new vertex(s) and edge(s)", hence $l_v(t) = \Phi_{A_v}(\{A_u, \{a\}\})$, where t is either an internal vertex of P_{new} or an edge of P_{new}. Lastly, again from the fact that Φ is compatible with $(D, \mathcal{X})$ and from step "update of labels of other vertex(s) and edge(s):

$$\ell_v(v_{att}) = F_\Phi(\ell_u(v_{att}), \mathcal{T}_{v_{att}}, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_{v_{att}} \cap A_u), \mathcal{A}_{v_{att}} \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_{v_{att}}).$$

Hence, at step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is a labeled partitioning tree of A_v. Therefore, the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$" guarantees that $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v. Thus, any element of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v.

We now show that any labeled q-branched partitioning tree with width not bigger than k for A_v is in $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Let $((T'_v, r'_v, \sigma'_v), \ell_v)$ be any q-branched partitioning tree with width not bigger than k for A_v. Let (T_u, r_u, σ_u) be (T'_v, r'_v, σ'_v) restricted to A_u. The partitioning tree (T_u, r_u, σ_u) is q-branched and its Φ-width is not bigger than k, since we only remove branches when restricting a partitioning tree and Φ is monotone. Thus, $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$.

Let v_{leaf} be the vertex of T'_v that corresponds to a. Since T_u is a subtree of T'_v, let $v_{att} \in V(T_u)$ be the vertex that is closest to v_{leaf} in T'_v and P_{new} be the path in T'_v joining v_{leaf} to v_{att}. Since $A_u = A_v \setminus \{a\}$, all internal vertices of P_{new} have degree two in T'_v. Hence, T'_v can be obtained from T_u in the step "update T_u into T_v" by attaching the vertex v_{leaf} to v_{att} and subdividing the edge e_{new} an amount of times equal to $|V(P_{new} \setminus \{v_{leaf}, v_{att}\})|$.

Therefore, let $((T_v, r_v, \sigma_v), \ell_v)$ be the labeled q-branched partitioning tree obtained from $((T_u, r_u, \sigma_u), \ell_u)$ with the Introduce Node procedure by adding a vertex v_{leaf} mapping a to v_{att} and subdividing $\{v_{att}, v_{leaf}\}$ an amount of times equal to $|V(P_{new} \setminus \{v_{leaf}, v_{att}\})|$. In other words, T_v and T'_v are isomorphic. Since the root of the tree does not change with the Introduce Node procedure, $r'_v = r_v = r_u$. Moreover, σ'_v and σ_v are the same, i.e. $\sigma'_v = \sigma_v = \sigma_u \cup (v_{att}, a)$. Therefore $((T'_v, r'_v, \sigma'_v), \ell'_v) = ((T_v, r_v, \sigma_v), \ell_v)$. Thus, for every q-branched partitioning tree $((T'_v, r'_v, \sigma'_v), \ell_v)$ with width not bigger than k for A_v there is an execution of process Introduce Node such that $((T'_v, r'_v, \sigma'_v), \ell_v) \in \mathrm{FSPT}_{\mathrm{k,q}}(v)$. □

4.5 Procedure Forget Node

Let v be a forget node of D, u be its child and $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be a full set of labeled q-branched partitioning trees of A_u with width at most k. Then procedure Forget Node consists of copying $\mathrm{FSPT}_{\mathrm{k,q}}(u)$. In other words, $\mathrm{FSPT}_{\mathrm{k,q}}(v) = \mathrm{FSPT}_{\mathrm{k,q}}(u)$.

Lemma 13. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be a forget node of D with a child u. The procedure Forget Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k for a forget node v of D.*

Since $A_v = A_u$ we have that $\mathrm{FSPT}_{\mathrm{k,q}}(v) = \mathrm{FSPT}_{\mathrm{k,q}}(u)$, therefore if v is a forget node this procedure produces a full set of labeled q-branched partitioning trees of A_u with width at most k for the node v.

4.6 Procedure Join Node

Let v be a join node of D, u and w its children and $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ a full set of labeled q-branched partitioning trees of A_u with width at most k and $\mathrm{FSPT}_{\mathrm{k,q}}(w)$ a full set of labeled q-branched partitioning trees of A_w with width at most k.

Let $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$. The goal is to merge "compatible" partitioning trees. We recall that, by definition of a join node of a nice decomposition, $X_u = X_w = X_v$. Hence, let $(T_u^r, r_u^r, \sigma_u^r)$ be (T_u, r_u, σ_u) restricted to X_v, i.e. the minimum subtree of T_u spanning all the leaves corresponding to elements of X_v (r_u^r is the vertex of T_u^r closest to r_u in T_u), and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_w, r_w, σ_w) restricted to X_v. Then, we do the following procedure for every pair of elements of $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$ such that T_u^r and T_w^r are isomorphic, $\sigma_u^r = \sigma_w^r$ and for every possible execution of step "Identifying T_u and T_w".

Identifying T_u and T_w: T_v is obtained by identifying all correspondent vertices of T_u^r and T_v^r. Then we remove from T_v the double edges resulting from the identification process. The root of T_v is obtained arbitrarily choosing an internal vertex of T_v. The mapping σ_v is obtained taking both mappings σ_u and σ_w, i.e. the leaves that are in T_u^r and T_w^r keep their correpondence to the elements of A. Since $X_u = X_w = X_v$, leaves that correspond to elements of X_v have the same mapping in σ_u and σ_w. Leaves that belong to $A_u \setminus A_w$ or $A_w \setminus A_u$ are only mapped by σ_u or σ_w respectively. Note that, at this point, (T_v, r_v, σ_v) is a partitioning tree of A_v.

Updating the labels: Let $(T_v^r, r_v^r, \sigma_v^r)$ be (T_v, r_v, σ_v) restricted to X_v. Let t_v be a vertex of T_v^r and $\mathcal{T}_{t_v}$ be the partition of X_v defined by t_v. Let t_u and t_w be the vertices of T_u^r and T_w^r, respectively, used to create t_v. Then, $\ell_v(t_v) \leftarrow H_\Phi(\ell_u(t_u), \ell_w(t_w), \mathcal{T}_{t_v})$. Let e_v be an edge of T_v^r and $\mathcal{T}_{e_v}$ the partition of X_v defined by e_v. Let e_u and e_w be its correspondent edges in T_u^r and T_w^r respectively. Then, $\ell_v(e_v) \leftarrow H_\Phi(\ell_u(e_u), \ell_w(e_w), \mathcal{T}_{e_v})$. For all other vertex (or edge) t of T_v: $\ell_v(t) \leftarrow \ell_u(t)$ if t is a vertex (or edge) of T_u or $\ell_v(t) \leftarrow \ell_v(t)$ if t is a vertex (or edge) of T_w.

Updating $\mathrm{FSPT}_{\mathrm{k,q}}(v)$**:** If $((T_v, r_v, \sigma_v), \ell_v)$ is q-branched, for every internal vertex $t \in V(T_v)$ we have $\ell_v(t) \leq k$ and for every edge $e \in E(T_v)$ we have $\ell_v(e) \leq k$ then $\mathrm{FSPT}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSPT}_{\mathrm{k,q}}(v) \cup \{((T_v, r_v, \sigma_v), \ell_v)\}$, otherwise $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ remains unchanged.

Lemma 14. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be a join node of D with a child u and a child w. The procedure Join Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k from the sets $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $\mathrm{FSPT}_{\mathrm{k,q}}(w)$ for the node v.*

Proof. Using the same scheme for the correctness of procedure Introduce Node, we first prove that all elements of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ described by the procedure Join Node are in fact labeled q-branched partitioning trees with width not bigger than k for A_v.

Let $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$, be such that (T_u, r_u, σ_u) restricted to X_u and (T_w, r_w, σ_w) restricted to X_w satisfy the Join Node restrictions. In other words, let $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_u, r_u, σ_u) and (T_w, r_w, σ_w) restricted to X_v respectively. Then, T_u^r and T_w^r are isomorphic and $\sigma_u^r = \sigma_w^r$.

The step "Identifying T_u and T_w" can be applied on $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$. Since $A_v = A_u \cup A_w$, $X_v = X_u = X_w$ and $(A_u \setminus X_u) \cap (A_w \setminus X_w) = \emptyset$, we have that (T_v, r_v, σ_v) obtained through step "Identifying T_u and T_w" is a partitioning tree of A_v.

For any internal vertex (or edge) of T_v let $\mathcal{A}_v$ be the partition of A_v that it defines. It remains to show that (T_v, r_v, σ_v) is q-branched, has Φ-width not bigger than k and that after the execution of the procedure Join Node all labels are correct. In other words, for all internal vertices (or edges) t of T_v, we prove that $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$.

In the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is q-branched. Lastly, we have that Φ is compatible with the nice decomposition $(D, \mathcal{X})$. Moreover, let $\mathcal{A}_t$, for any internal vertex (or edge) t of T_v be the partition of A_v defined by t. Then, from the fact that $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$ are labeled partitioning trees, $\ell_u(t_u) = \Phi_{A_u}(\mathcal{A}_{t_u} \cap A_u)$ for any internal vertex (or edge) t_u of T_u and $\ell_w(t_w) = \Phi_{A_w}(\mathcal{A}_{t_w} \cap A_w)$ for any internal vertex (or edge) t_w of T_w. Therefore, from the description of the Join Node procedure for every internal node (or any edge) of T_v:

$$\ell_v(t) = H_\Phi(\ell_u(t), \ell_w(t), \mathcal{T}_t) = H_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \Phi_{A_w}(\mathcal{A}_t \cap A_w), \mathcal{A}_t \cap X_v) = \Phi_{A_v}(\mathcal{A}_t)$$

Hence, at step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is a labeled partitioning tree of A_v. Therefore, the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$" guarantees that $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v.

We now show that any labeled q-branched partitioning tree with width not bigger than k for A_v is in $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Let $((T'_v, r'_v, \sigma'_v), \ell_v)$ be any q-branched partitioning tree with width not bigger than k for A_v.

Let (T_u, r_u, σ_u) be (T'_v, r'_v, σ'_v) restricted to A_u and (T_w, r_w, σ_w) be (T'_v, r'_v, σ'_v) restricted to A_w. The partitioning trees (T_u, r_u, σ_u) and (T_w, r_w, σ_w) are, by definition of restriction, q-branched and their Φ-width is not bigger than k, thus $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$.

Let $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_u, r_u, σ_u) and (T_w, r_w, σ_w) restricted to X_v respectively. Since $X_u = X_w = X_v$ we have that $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ are such that T_u^r and T_w^r are isomorphic and $\sigma_u^r = \sigma_w^r$. Therefore, the Join Node procedure is applied to (T_u, r_u, σ_u) and (T_w, r_w, σ_w).

Therefore, let $((T_v, r_v, \sigma_v), \ell_v)$ be the labeled q-branched partitioning tree obtained from $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$ with the Join Node procedure. Clearly, from the "Identifying T_u and T_v" step, $T_v = T_v'$ and $\sigma_v = \sigma_v'$. Since the procedure Join Node chooses an arbitrary internal vertex as the root of the partitioning tree, there is an execution of this step where r_v is chosen as the root of T_v. Therefore, $((T_v', r_v', \sigma_v'), \ell_v') = ((T_v, r_v, \sigma_v), \ell_v)$. □

4.7 Remarks on Width Measures

Given a graph $G = (V, E)$ and a nice decomposition $(D, \mathcal{X})$ of E or V. Then, with $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ where r is the root of $(D, \mathcal{X})$, it is possible to answer if the (tree, path, branch, linear, cut, carving) width of G is less or equal than k. For that, we iterate among all $((T, r, \sigma), \ell) \in \mathrm{FSPT}_{\mathrm{k,q}}(r)$ and search for one that obeys the "structural" restrictions given by the desired width, for example, partitioning trees for the branch width are such that every internal vertex has degree three, hence it is necessary to search $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ for a labeled partitioning tree that respects such restriction. In the case that there are no labeled partitioning trees with the "structural" restrictions given, then the desired width of G is bigger than k, otherwise we found a partitioning tree that proves that the desired width of G is not bigger than k.

The following sections of the paper address the fact that the number of elements of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is possibly infinite. In Section 5 we show how to store $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ in an efficient manner, by only storing "compressed" representatives for each "class" of partitioning tree. Lastly, in Section 6, we show how to manipulate these compressed representatives in order to design an algorithm for this problem. This manipulation is a direct extension of the procedures Start Node, Introduce Node, Join Node and Forget Node when applied to "compressed" representatives of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$.

5 Good Representatives of Partitioning Trees

In this section we outline the ideas used in order to improve the space necessary to store the q-branched partitioning trees of a node in the nice decomposition of A. In other words, in this section, we reuse a method for "compressing" path decompositions and tree decomposition in [BK96] this time applied to q-branched partitioning trees and partitioning functions. A "compression" of the set $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for a node v of the nice decomposition of A is such that the size of this compression is bounded by q, the Φ-width and the width of the nice decomposition of A. Hence, it does not depend on the size of A. Intuitively, this is achieved by keeping only "good" representatives, a.k.a. characteristics, for each q-branched partitioning tree with Φ-width at most k of A.

5.1 Labeled Paths

Let Φ be a monotone partition function. As stated in Section 4.1, any partitioning tree (T, σ) can be viewed as a labeled graph, where any $v \in V(T)$ is labeled with $\Phi(\mathcal{T}_v)$ and any $e \in E(T)$ is labeled with $\Phi(\mathcal{T}_e)$. Because Φ is monotone, the label of an edge is not bigger than the label of its endpoints. In this section, we detail operations over labeled paths, that will serve as the basis of manipulating partitioning trees in the forthcoming sections.

A *labeled path* P is a path $(v_0, v_1, \ldots, v_n)$ where any vertex v_i is labeled with an integer $\ell(v_i)$, and any edge $e_i = \{v_{i-1}, v_i\}$ with an integer $\ell(e_i)$ such that the label of any edge is less or equal to the label of any of its endpoints.

A vertex $v_i \in V(P)$ or an edge $\{v_i, v_{i+1}\}$ is smaller than $v_j \in V(P)$ if $i < j$. Similarly v_i (resp., $\{v_i, v_{i+1}\}$) is smaller than $\{v_j, v_{j+1}\}$ if $i < j$. We define $\max(P)$ as the maximum integer labeling an edge or a vertex of P. Similarly, we define $\min(P)$.

Let $e = \{u, v\}$ be an edge in which we do a subdivision, let $e_1 = \{u, x\}$ and $e_2 = \{x, v\}$ be the edges in the resulting path and x be the vertex created, then $\ell(e_1) = \ell(e_2) = \ell(x) = \ell(e)$, i.e. edges and the vertex resulting from the subdivision are labeled with $\ell(e)$. An *extension* of a labeled path P is any path obtained by subdividing some edges of P an arbitrary number of times. Let P^* be an extension of P. The *originator* of an edge $e^* \in E(P^*)$ is the edge $e \in E(P)$ such that e^* is obtained in P^* by the subdivision of e. Similarly, the *originator* of a vertex $v^* \in E(P^*)$ is the correspondent vertex of P, if v^* is not the result of a subdivision, or is the edge $e \in E(P)$ that was subdivided to create v^*.

For any function $F : \mathbb{N} \to \mathbb{N}$, let $F(P)$ denote the path $(v_0, v_1, \ldots, v_n)$ where any label ℓ has been replaced by $F(\ell)$. If $P = (v_1, \ldots, v_n)$ and $Q = (w_1, \ldots, w_m)$ are two labeled paths with a common end, $v_n = w_1$, and vertex disjoint otherwise, their *concatenation* $P \odot Q$ is the labeled path $(v_1, \ldots, v_n = w_1, w_2, \ldots, w_m)$.

5.1.1 Contraction of a labeled path

In this section, we define an operation on labeled paths that will be widely used in the next sections. This operation is used to compress the size of a q-branched partitioning tree. For this purpose, we revisit the notion of *typical sequence* of a sequence of integers [BK96]. Roughly, the goal of the following operation is to contract some edges and vertices of P that are not "necessary" to remember the variations of the sequence $(\ell(v_0), \ell(e_1), \ell(v_1), \ldots, \ell(e_n), \ell(v_n))$.

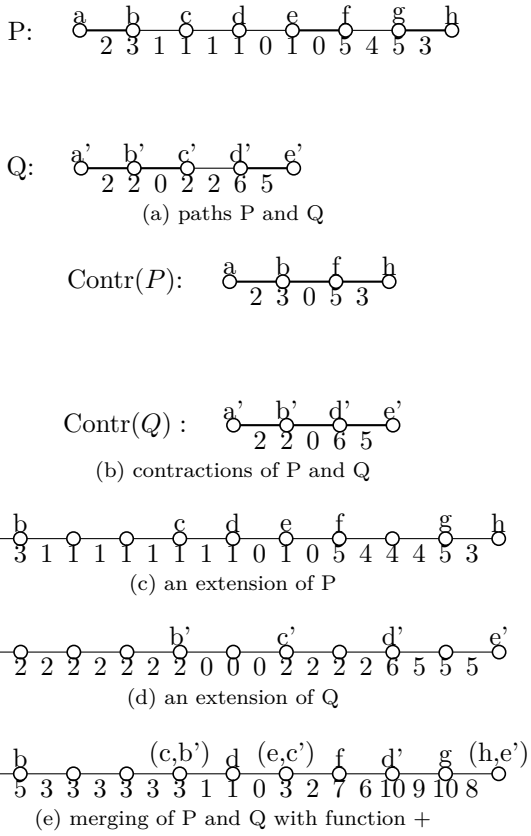
First, let us recall the definition of the typical sequence of a sequence of integers [BK96]. Let $S = (s_i)_{i \leq 2n-1}$ be a sequence of integers. Its typical sequence $\tau(S)$ is obtained by iterating the following operations while it is possible: (1) if there is $i < |S|$ such that $s_i = s_{i+1}$, remove s_{i+1} from S, and (2) if there are $i < j-1 < |S|$, and either, for any $i \leq k \leq j$, $s_i \leq s_k \leq s_j$, or, for any $i \leq k \leq j$, $s_i \geq s_k \geq s_j$, remove s_k from S for any $i < k < j$. Note that the order in which the operations are executed is not relevant, therefore $\tau(S)$ is uniquely defined.

The *contraction* $\mathrm{Contr}(P)$ is the path obtained from $P = (v_0, \ldots, v_n)$ with same ends by contracting some edges and vertices with the following operations until no more vertices or edges can be removed from the path, let $e_i = \{v_{i-1}, v_i\}$:

Operation 1: There exists $0 < i \leq k \leq n$ such that $\forall_{i \leq j \leq k} \ell(e_i) \leq \ell(e_j) \leq$

$\ell(v_k)$ and $\forall_{i\leq j<k}\ell(e_i) \leq \ell(v_j) \leq \ell(v_k)$, then P becomes $(v_0, \ldots, v_{i-1}, v_k, \ldots, v_n)$ and $\ell(\{v_{i-1}, v_k\}) = \ell(e_i)$.

Operation 2: There exists $0 \leq i < k \leq n$ such that $\forall_{i<j\leq k}\ell(v_i) \geq \ell(e_j) \geq \ell(e_k)$ and $\forall_{i\leq j<k}\ell(v_i) \geq \ell(v_j) \geq \ell(e_k)$, then P becomes $(v_0, \ldots, v_i, v_k, \ldots, v_n)$ and $\ell(\{v_i, v_k\}) = \ell(e_k)$.



(a) paths P and Q

(b) contractions of P and Q

(c) an extension of P

(d) an extension of Q

(e) merging of P and Q with function +

Figure 2: Labeled paths: contraction, extension and merging. The labels of first and last vertex are omitted, since they do not change with any operation.

It is important to note that any $v \in V(\text{Contr}(P))$ (resp. $e \in E(\text{Contr}(P))$) *represents* a unique $v^* \in V(P)$ (resp. $e^* \in E(P)$), i.e. the vertex (resp. edge) in $V(P)$ (resp. $E(P)$) that originated v (resp. e) during the contraction operation. By this definition, if x represents x^* then $\ell(x) = \ell(x^*)$.

Figure 2 represents two labeled paths P and Q. The vertices of $\text{Contr}(P)$ and $\text{Contr}(Q)$ are named as the vertices they represent in P and Q. We also illustrate an extension P^* of P and an extension Q^* of Q.

In the following, let e'_i be the edge $\{v'_{i-1}, v'_i\}$. The crucial property of $\text{Contr}(P) = (v_0 = v'_0, v'_1, \ldots, v'_{p-1}, v'_p = v_n)$ is that the sequence $S' = (\ell(e'_1), \ell(v'_1), \ldots, \ell(e'_{p-1}), \ell(v'_{p-1}), \ell(e'_p))$ is "almost" the typical sequence $\tau(S)$ of $S = (\ell(e_1), \ell(v_1), \ldots, \ell(e_n))$. More precisely,

- if $\ell(e'_1) \neq \ell(v'_1)$ and $\ell(e'_p) \neq \ell(v'_{p-1})$, then $S' = \tau(S)$;

- if $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) \neq \ell(v'_{p-1})$, then $S' = \ell(e'_1) \cdot \tau(S)$;
- if $\ell(e'_1) \neq \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, then $S' = \tau(S) \cdot \ell(e'_p)$;
- if $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, then $S' = \ell(e'_1) \cdot \tau(S) \cdot \ell(e'_p)$.

Lemma 15. *Let P be a labeled path.*
1. $\min(\mathrm{Contr}(P)) = \min(P)$ *and* $\max(\mathrm{Contr}(P)) = \max(P)$.
2. *Given a labeled path P with $\max(P) \leq k$, then the number of edges, or size, of $\mathrm{Contr}(P)$ is at most $2k + 3$.*

Proof. Assume that $P = (v_1, v_2, \ldots, v_n)$. We recall that the labels of the edges are not bigger than the labels of their endpoints. Note that, the contraction operations do not change the labels of vertices or edges of P, they simply remove some vertices or edges of P. Hence, $\min(\mathrm{Contr}(P)) \geq \min(P)$ and $\max(\mathrm{Contr}(P)) \leq \max(P)$.

Assume that there exists P' such that P' is obtained after one contraction operation is applied to P and that $\max(P') < \max(P)$. W.l.o.g. assume that the contraction operation was applied between edge the edge $e_i = \{v_{i-1}, v_i\}$ and the vertex v_j, $j > i$. In other words, $P' = (v_1, \ldots, v_{i-1}, v_j, \ldots, v_n)$. Since, $\max(P') < \max(P)$ we must have removed a vertex with a big label, That is, there is $v_l \in V(P)$, $i \leq l \leq j$, such that $\ell(v_l) > \ell(v_j)$. Therefore, by definition of a contraction operation, we are not allowed to do a contraction operation between e_i and v_j. Hence, $\min(P) = \min(P')$.

Assume that there exists P' such that P' is obtained after one contraction operation is applied to P and that $\min(P') > \min(P)$. W.l.o.g. assume that the contraction operation was applied between edge the edge $e_i = \{v_{i-1}, v_i\}$ and the vertex v_j, $j > i$. In other words, $P' = (v_1, \ldots, v_{i-1}, v_j, \ldots, v_n)$. Since, $\min(P') < \min(P)$ we must have removed an edge with small label, That is, there is $e_l \in E(P)$, $i + 1 \leq l \leq j$, such that $\ell(e_l) < \ell(e_i)$. Therefore, by definition of a contraction operation, we are not allowed to do a contraction operation between e_i and v_j. Hence, $\min(P) = \min(P')$.

Let $P = (v_0, v_1, \ldots, v_n)$, $e_i = \{v_{i-1}, v_i\}$ and $S = (\ell(e_1), \ell(v_1), \ell(e_2), \ell(v_2), \ldots, \ell(v_n))$. We have that the number of edges plus the number of vertices of $\mathrm{Contr}(P) = (v'_0, v'_1, \ldots, v'_p)$ is at most the size of $\tau(S)$ plus two, since it is possible that $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, where $e'_i = \{v'_{i-1}, v'_i\}$.

Therefore, we have that the number of edges in $\mathrm{Contr}(P)$ is not bigger than $2k+3$, since the number of elements in $\tau(S)$ is not bigger than $2k+1$ [Bod96]. □

Lemma 16. *Let P and Q be two labeled paths. Let P^* be any extension of P.*
1. $\mathrm{Contr}(P^*) = \mathrm{Contr}(P) = \mathrm{Contr}(\mathrm{Contr}(P))$.
2. *Let $F : \mathbb{N} \to \mathbb{N}$ be any strictly increasing function. $F(\mathrm{Contr}(P)) = \mathrm{Contr}(F(P))$.*
3. $\mathrm{Contr}(P \odot Q) = \mathrm{Contr}(\mathrm{Contr}(P) \odot \mathrm{Contr}(Q))$.

Proof. 1. From the definition of "Contr" we have $\mathrm{Contr}(P) = \mathrm{Contr}(\mathrm{Contr}(P))$.

Let P' be obtained from P by subdividing one edge $e = \{u, v\}$ a k times resulting in $e_1 = \{u, t_1\}$, $e_2 = \{t_1, t_2\}$, $e_3 = \{t_2, t_3\}$, $\ldots$, $e_k = \{t_{k-1}, t_k\}$ and $e_{k+1} = \{t_k, v\}$. Then, for each $1 \leq i \leq k$, the labels of e_i and t_i are given by $\ell(e)$ and $\ell(e_{k+1}) = \ell(e)$, i.e. P' is the extension of P where e is subdivided k times. Then, a contraction operation can be applied between e_1 and v, $\ell(e_1) \leq \ell(t_1) = \ell(e_2) = \ell(t_2) = \cdots = \ell(t_k) = \ell(e_{k+1}) \leq \ell(v)$. The result of this contraction on path P' is P.

Then, by induction on the number of edges of P that were subdivided. Clearly, the result holds if zero edges are subdivided. Let P^* be an extension of P that subdivides at most $i \geq 0$ different edges. The above reasoning shows that it is possible to contract edges and vertices that are created through a subdivision of a single edge, i.e., if $e = \{u, v\}$ is subdivided a finite amount of times, then it is possible to apply a contraction operation on the edge created by the subdivision that has u as endpoint and v. Let P' be obtained from P^* by applying a contraction operation between v and e_1. By induction hypothesis, $\text{Contr}(P') = \text{Contr}(P)$, since $\text{Contr}(P^*) = \text{Contr}(P')$ we got the result. That is, $\text{Contr}(P^*) = \text{Contr}(P)$.

2. In what follows we abuse the notation of a path to include the set of edges of P, i.e. a path $P = (v'_1, \ldots, v'_r)$ becomes the path $P = (v_1, e_2, v_3, e_4, \ldots, e_{r-1}, v_{2r-1})$ where $v_{2i-1} = v'_i$ and $e_i = \{v_{i-1}, v_{i+1}\}$.

Let P be equal to $(v_1, e_2, v_3, \ldots, v_i)$ and P^f be the labeled path obtained through P by applying F, that is, $F(P)$. We show that $F(\text{Contr}(P)) = \text{Contr}(F(P))$ by showing that a contraction operation can be performed in P if and only if it can be performed in $F(P)$.

W.l.o.g. assume that it is possible to do a contraction operation between the edge e_i and vertex v_j, $j > i$, in P. Hence, for all e_k and v_k, $i < k < j$ we have that $\ell(e_i) \leq \ell(e_k) \leq \ell(v_j)$ and $\ell(e_i) \leq \ell(v_k) \leq \ell(v_j)$. Since, F is strictly increasing, we have that for all e_k and v_k, $i < k < j$, $F(\ell(e_i)) \leq F(\ell(e_k)) \leq F(\ell(v_j))$ and $F(\ell(e_i)) \leq F(\ell(v_k)) \leq F(\ell(v_j))$. Therefore, in $F(P)$ it is possible to apply a contraction operation between e_i and v_j.

Similarly, assume that it is possible to do a contraction operation between the edge e_i and vertex v_j, $j > i$, in $F(P)$. Hence, for all e_k and v_k, $i < k < j$ we have that $F(\ell(e_i)) \leq F(\ell(e_k)) \leq F(\ell(v_j))$ and $F(\ell(e_i)) \leq F(\ell(v_k)) \leq F(\ell(v_j))$. Since, F is strictly increasing, we have that for all e_k and v_k, $i < k < j$, $\ell(e_i) \leq \ell(e_k) \leq \ell(v_j)$ and $\ell(e_i) \leq \ell(v_k) \leq \ell(v_j)$. Therefore, in P it is possible to apply a contraction operation between e_i and v_j.

Hence, the paths $\text{Contr}(P) = \{v_1^c, \ldots, v_n^c\}$ and $\text{Contr}(F(P)) = \{v_1^f, \ldots, v_n^f\}$ are composed of the same sequence of vertices and edges. That is, for all $1 \leq i \leq n$ we have that v_i^c and v_i^f represent the same vertex in P. Therefore, $F(\text{Contr}(P)) = \text{Contr}(F(P))$.

3. Comes directly from the fact that the order of contractions does not change the path obtained by applying "Contr" to a path. □

Let $P = (v_0, \ldots, v_n)$, a simple property of $\text{Contr}(P)$ is that: if v_j is a vertex with a representative in $\text{Contr}(P)$ we have that $\text{Contr}(P) = \text{Contr}((v_0, \ldots, v_j)) \odot \text{Contr}((v_j, \ldots, v_n))$.

A scheme of Lemma17 can be found in Figure 3.

Lemma 17. *Let $P = (v_0, \ldots, v_n)$ be a labeled path and $\text{Contr}(P) = (v_0^c, \ldots, v_p^c)$. Let $i \leq p$. Let $P^c = (v_0^c, \ldots, v_{i-1}^c, x, v_i^c, \ldots, v_p^c)$ be the extension of $\text{Contr}(P)$ obtained by subdividing once the edge $e_i^c = \{v_{i-1}^c, v_i^c\} \in \text{Contr}(P)$. Let $e_i^* = \{v_{j-1}, v_j\}$ be an edge of P represented by e_i^c, and $P' = (v_0, \ldots, v_{j-1}, y, v_j, \ldots, v_n)$ be the extension of P obtained by subdividing once e_i^*. Let $P_1^c = (v_0^c, \ldots, x)$, $P_2^c = (x, \ldots, v_p^c)$, $P_1 = (v_0, \ldots, y)$ and $P_2 = (y, \ldots, v_n)$. Then, $\text{Contr}(P_1) = \text{Contr}(P_1^c)$ and $\text{Contr}(P_2) = \text{Contr}(P_2^c)$.*

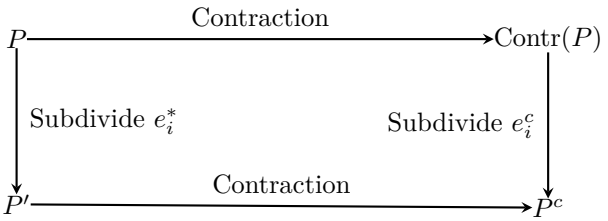


Figure 3: Scheme of Lemma 17.

Proof. Let v_a be the vertex of P_1 represented by v_{i-1}^c in Contr(P) and v_b the vertex of P_2 represented by v_i^c in Contr(P). Therefore, from the definition of "Contr", Contr($(v_0, \ldots, v_a)$) = Contr($(v_0^c, \ldots, v_{i-1}^c)$) and Contr($(v_b \ldots, v_n)$) = Contr($(v_i^c, \ldots, v_p^c)$).

Since $\ell(\{v_{j-1}, y\}) = \ell(\{y, v_j\}) = \ell(\{v_{i-1}^c, x\}) = \ell(\{x, v_i^c\}) = \ell(\{v_{i-1}^c, v_i^c\}) = \ell(\{v_{j-1}, v_j\})$, we have that any contraction operation applied between v_a and $\{v_{j-1}, v_j\}$ or between $\{v_{j-1}, v_j\}$ and v_b in P can also be applied in P' between v_a and $\{v_{j-1}, y\}$ or, in the second case, between $\{y, v_j\}$ and v_b.

Therefore, Contr($(v_a, \ldots, y)$) = Contr((v_{i-1}^c, x)) and Contr($(y, \ldots, v_b)$) = Contr((x, v_i^c)). Then, from item 3 of Lemma 16 we have:

$$\begin{aligned}\text{Contr}(P_1) &= \text{Contr}(\text{Contr}((v_0, \ldots, v_a)) \odot \text{Contr}((v_a, \ldots, y))) \\ &= \text{Contr}(\text{Contr}((v_0^c, \ldots, v_{i-1}^c)) \odot \text{Contr}((v_{i-1}^c, x))) \\ &= \text{Contr}(P_1^c)\end{aligned}$$

The proof for Contr(P_2) = Contr(P_2^c) is similar an thus omitted. □

5.1.2 Merging of labeled paths

Now, we present an operation that merges two labeled paths P and Q with common ends and vertex-disjoint otherwise. This operation is used to aid in the computation of the full set of a join node in the algorithm. The assumption that the paths have common ends is a reflection of how this operation is used by the algorithm of Section 6, but it is not intrinsically necessary.

A *merging* $M = (m_1, \ldots, m_k)$ of two labeled paths $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_m)$ under a function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a path obtained by constructing extensions $P^e = (p_1^e, \ldots, p_k^e)$ and $Q^e = (q_1^e, \ldots, q_k^e)$ of P and Q such that P^e and Q^e have the same length, $k \leq nm$. Then set $\ell(\{m_{i-1}, m_i\}) = F(\ell(\{p_{i-1}^e, p_i^e\}), \ell(\{q_{i-1}^e, q_i^e\}))$, for all $2 \leq i \leq k$, and $\ell(m_i) = F(\ell(p_i^e), \ell(q_i^e))$, for all $1 \leq i \leq k$. Note that, the maximum size of the merging, $k \leq mn$, is big enough to merge each edge on one path to each edge on the other path.

Figure 2 represents a merging of P and Q using the function $F : (x, y) \to x + y$. When merging P and Q, we assume they have same ends, i.e. $p_1 = q_1$ and $p_n = p_m$.

Let P and Q be two labeled paths and $M = (m_1, \ldots, m_k)$ be a merging of P and Q under a function F. Let $P^e = (p_1^e, \ldots, p_k^e)$ and $Q^e = (q_1^e, \ldots, q_k^e)$ be the extensions of P and Q used to create M.

We say that a vertex $m_i \in V(M)$ matches $p_i^e \in V(P^e)$ and $q_i^e \in V(Q^e)$. Similarly, we say that an edge $\{m_i, m_{i+1}\} \in E(M)$ matches $\{p_i^e, p_{i+1}^e\} \in E(P^e)$

and $\{q_i^e, q_{i+1}^e\} \in E(Q^e)$.

Let m be a vertex or edge of M such that m matches p^e and q^e in P^e and Q^e respectively, then $\text{Orig}(m)$ is the pair (p, q) such that $p \in V(P) \cup E(P)$ is the originator, the vertex or edge of P that originated p^e, of p^e and $q \in V(Q) \cup E(Q)$ is the originator of q^e.

We now list some simple but useful properties of "Orig":

- if e is an edge of M, then $\text{Orig}(e) = (p, q)$ is such that $p \in E(P)$ and $q \in E(Q)$;
- if v is a vertex of M and $\text{Orig}(v) = (p, q)$ is such that $p \in E(P)$ and $q \in E(Q)$ then, let e be any edge of M with v as extremity, $\text{Orig}(e) = (p, q)$. That is, if v is obtained by merging two vertices that were obtained from the subdivision of an edge, then the edges incident to v are also originated from the same edges;
- for every vertex $p \in V(P)$ (resp. $q \in V(Q)$) there is only one $v \in V(M)$ such that $\text{Orig}(v) = (p, x)$ (resp. $\text{Orig}(v) = (x, q)$), where x is either a vertex or edge of Q (resp. P). That is, since vertices of P and Q cannot be subdivided, they can only originate one vertex in M;
- if $M = (m_1, \ldots, m_k)$, then $\text{Orig}(m_1) = (p_1, q_1)$ and $\text{Orig}(m_k) = (p_n, q_m)$ where p_1 and p_n are the extremities of P and q_1 and q_m the extremities of Q.

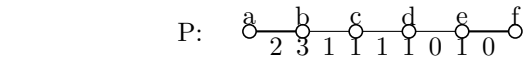
Let P and Q be two labeled paths and M a merging of P and Q under a function F. Let P^c (resp. Q^c) be path obtained from P (resp. Q) after some, possibly zero, contraction operations are applied to P (resp. Q). Let $M^c = (m_1^c, \ldots, m_k^c)$ be a merging of P^c and Q^c under the same function F.

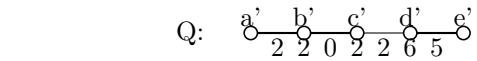
Roughly, we say that a merging M of P and Q *respects* a merging M^c of P^c and Q^c if the vertices and edges in M^c are obtained by matching "correspondent" vertices or edges in P and Q. Figure 4 shows an example of paths P, Q, P^c, Q^c, a merging M^c of P^c and Q^c and a merging M of P and Q that respects M^c.

For any vertex or edge m^c of M^c, if $\text{Orig}(m^c) = (p^c, q^c)$, then let $p^c \in V(P^c) \cup E(P^c)$ and $q^c \in V(Q^c) \cup E(Q^c)$ be the representatives of $p \in V(P) \cup E(P)$ and $q \in V(Q) \cup E(Q)$ respectively. Formally, we say that M *respects* M^c if the two following conditions are met:

- for all vertices $m \in V(M)$ with $\text{Orig}(m) = (p, q)$ such that p has a representative p^c in P^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\text{Orig}(m^c) = (p^c, q^c)$, where q^c is any vertex or edge of Q^c;
- for all vertices $m \in V(M)$ with $\text{Orig}(m) = (p, q)$ such that q has a representative q^c in Q^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\text{Orig}(m^c) = (p^c, q^c)$, where p^c is any vertex or edge of P^c;
- for all vertices $m \in V(M)$ with $\text{Orig}(m) = (p, q)$ such that such that p has a representative p^c in P^c and q has a representative q^c in Q^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\text{Orig}(m^c) = (p^c, q^c)$;
- for all edges $e \in E(M)$ with $\text{Orig}(e) = (p, q)$ such that p has a representative p^c in P^c, we have that there is an edge $e^c \in E(M^c)$ such that $\text{Orig}(e^c) = (p^c, q^c)$, where q^c is any vertex or edge of Q^c;

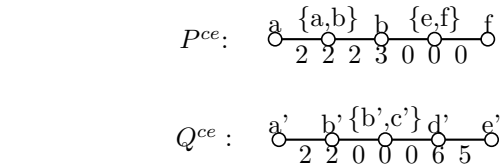
- for all edges $e \in E(M)$ with $\mathrm{Orig}(e) = (p, q)$ such that q has a representative q^c in Q^c, we have that there is an edge $e^c \in V(M^c)$ such that $\mathrm{Orig}(e^c) = (p^c, q^c)$, where p^c is any vertex or edge of P^c;
- for all edges $e \in E(M)$ with $\mathrm{Orig}(e) = (p, q)$ such that such that p has a representative p^c in P^c and q has a representative q^c in Q^c, we have that there is a an edge $e^c \in V(M^c)$ such that $\mathrm{Orig}(m^c) = (p^c, q^c)$.



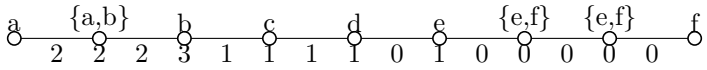


(a) paths P and Q

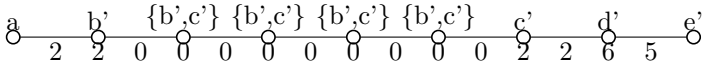
(b) P^c and Q^c



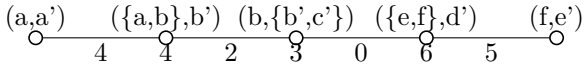
(c) extensions of P^c and Q^c



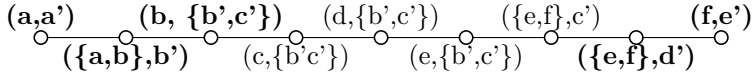
(d) extension of P



(e) extension of Q



(f) M^c: merging of P^{ce} and Q^{ce} under function +



(g) M: merging of P and Q with function + respecting M^c

Figure 4: A representation of a merging of P and Q that respects the merging of its contractions. Only originators of vertices are shown to avoid overloading the figure. Note that, for each originator of a vertex of M^c there is a vertex of M that has the same originator. Moreover, these originators appear in the same order in both paths M^c and M.

Lemma 18. *Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ strictly increasing in both coordinates. Let P and Q be labeled paths. Let K_p and K_q be subsets of vertices of P and Q*

respectively. Let P^c be obtained from P by applying some contraction operations, but without contracting any vertex in K_p. Similarly, let Q^c be obtained from Q by applying some contraction operations, but without contracting any vertex in K_q.

Let M^c be a merging of P^c and Q^c under F and M be a merging of P and Q under F that respects M^c.

Let r be the biggest integer such that $M^c = M_1^c \odot \cdots \odot M_r^c$, where, for all $i < r$, the common end between M_i^c and M_{i+1}^c is a vertex $v_i^c \in V(M^c)$ such that $\mathrm{Orig}(v_i^c) = (p_i^c, q_i^c)$ where either the vertex represented by p_i^c is in K_p or the vertex represented by q_i^c is in K_q.

Let $M = M_1 \odot \cdots \odot M_r$, where, for all $i < r$, the common end between M_i and M_{i+1} is a vertex $v_i \in V(M)$ such that $\mathrm{Orig}(v) = (p_i, q_i)$ where either $p_i \in K_p$ or $q_i \in K_q$.

Then, for all $1 \leq i \leq r$ we have $\mathrm{Contr}(M_i) = \mathrm{Contr}(M_i^c)$.

Proof. In order to prove that $\mathrm{Contr}(M_i) = \mathrm{Contr}(M_i^c)$ for all $1 \leq i \leq r$, we prove that we can obtain M_i^c from M_i with some contraction operations.

More precisely, let $M_i = (m_1, \ldots, m_h)$ and $M_i^c = (m_1^c, \ldots, m_{r'}^c)$. Consider a vertex $m^c \in V(M_i^c)$ and an edge $e^c \in E(M_i^c)$ such that m^c is an extremity of e^c. Let $\mathrm{Orig}(m^c) = (p^c, q^c)$ and let $\mathrm{Orig}(e^c) = (p'^c, q'^c)$. Since M respects M^c, there is $m \in V(M_i)$ such that $\mathrm{Orig}(m) = (p, q)$ with p and q being represented by p^c and q^c respectively and there is $e \in E(M_i)$ such that $\mathrm{Orig}(e) = (p', q')$ with p' and q' being represented by p'^c and q'^c respectively. Choose m and e such that the amount of internal vertices on the subpath of M from m to the extremity of e that is closest to m is the biggest. That is, if e "appears" after m in the path M, then m is the first vertex of M such that $\mathrm{Orig}(m) = (p, q)$ and e is the last edge of M such that $\mathrm{Orig}(e) = (p', q')$.

We show that, in M_i, if m is not an extremity of e, then it is possible do a contraction operation between m and e. In what follows assume that m is not an extremity of e in M.

Since p^c and p'^c are the originators of m^c and e^c respectively, either they are the same, meaning that m^c and e^c are originated from the subdivision of an edge of P^c, or they are different, meaning that m^c originated from a vertex (or edge) of P^c and e^c originated from an edge (or vertex) of P^c. In the case that p^c and p'^c are different, since m^c is an extremity of e^c in M^c, it means that either p^c is a vertex that is an extremity of p'^c in P^c or that p^c is an edge with p'^c as extremity in P^c. Hence, if $p^c \neq p'^c$, then, since p^c represents p and p'^c represents p', it is possible to contract all vertices and edges between p and p' in P.

With a similar reasoning we have that, if $q^c \neq q'^c$, then it is possible to contract all vertices and edges between q and q' in Q.

There are four cases to consider: $p^c \neq p'^c$ and $q^c \neq q'^c$; $p^c = p'^c$ and $q^c \neq q'^c$; $p^c \neq p'^c$ and $q^c = q'^c$; $p^c = p'^c$ and $q^c = q'^c$.

1. $p^c \neq p'^c$ and $q^c \neq q'^c$: then, in P (resp. in Q), it is possible to contract all vertices between p and p' (resp. q and q'). Then, from the fact that H is strictly increasing in both coordinates, it is possible to contract all vertices between m and e in M.

   Formally, w.l.o.g. assume that $p^c \neq p'^c$ be such that $\ell(p^c) \geq \ell(p'^c)$ and that $q^c \neq q'^c$ be such that $\ell(q^c) \geq q'^c$. Then, in P, all vertices or edges x^p

between p and p' are such that $\ell(p') \leq \ell(x^p) \leq \ell(p)$ and, in Q, all vertices or edge x^q between q and q' are such that $\ell(q') \leq \ell(x^q) \leq \ell(q)$. Therefore, since H is strictly increasing in both coordinates, $H(\ell(p'), \ell(q')) \leq H(\ell(x^p), \ell(x^q)) \leq H(\ell(p), \ell(q))$ for all vertices or edges x^p and x^q that are between p and p' in P and between q and q' in Q respectively. Hence, in M, it is possible to contract all vertices between m and e.

2. $p^c = p'^c$ and $q^c \neq q'^c$: then, in Q, it is possible to contract all vertices between q and q'. Since $p^c = p'^c$, it means that $p = p'$, therefore the vertex m, the edge e an all vertices or edge in between m and e are obtained from a subdivision of the same edge p in P. Then, from the fact that H is strictly increasing in both coordinates and the fact the first coordinate of H is the same when applying to all vertices and edges on the path between m and e, it is possible to contract all vertices between m and e in M.

   Formally, w.l.o.g. assume that $p^c = p'^c$ and that $q^c \neq q'^c$ be such that $\ell(q^c) \geq q'^c$. Then, in Q, all vertices or edge x^q between q and q' are such that $\ell(q') \leq \ell(x^q) \leq \ell(q)$. Therefore, since H is strictly increasing in both coordinates, $H(\ell(p'), \ell(q')) \leq H(\ell(p'), \ell(x^q)) \leq H(\ell(p'), \ell(q))$ for all vertices or edges x^q that are between q and q' in Q. Hence, in M, it is possible to contract all vertices between m and e.

3. $p^c \neq p'^c$ and $q^c = q'^c$: this case is similar to the case ($p^c = p'^c$ and $q^c \neq q'^c$) and thus omitted.

4. $p^c = p'^c$ and $q^c = q'^c$: this means that m, e and all vertices and edges of M between m and e where obtained from the subdivision of an edge p in P and an edge q in Q, hence they all have the same label which is given by $H(\ell(p), \ell(q))$. Therefore, it is possible to do a contraction operation between m and e in M.

Then, let M'_i be obtained from M_i by applying the above reasoning for each edge $e^c \in E(M^c)$ and with both its extremities. The resulting path M'_i is M_i^c. Since $\mathrm{Contr}(M'_i) = \mathrm{Contr}(M_i^c)$ and $\mathrm{Contr}(M_i) = \mathrm{Contr}(M'_i)$, we have the result. □

5.2 Characteristics - Good Representatives

In this section we introduce the notion of "characteristic" of a partitioning tree. The idea behind a characteristic is that, in order to compute a partitioning tree for a node v in the nice decomposition $(D, \mathcal{X})$, the "only" important information is given by elements of X_v and the "structure" of the partitioning tree. Hence, it is possible to "forget" elements of $A_v \setminus X_v$ from the partitioning trees for v.

5.2.1 Restriction of a partitioning tree

Let (T', r', σ') be partitioning tree of a set A and Φ_A a monotone partition function over A. We assume that T' is not restricted to an edge, and r' is not a leaf of T' to avoid unnecessary simple cases. Therefore, the corpse $\mathrm{cp}(T')$ (T' without its leaves) can be rooted in r'. Let $B \subseteq A$, the *restriction* $\mathrm{Char}((T', r', \sigma'), B)$ *of* (T', r', σ') *to* B is composed of the following structures:

- a rooted partitioning tree (T, r, σ) of B;

- an integer *dist* used to remember the number of branching nodes between r' and r;
- a subset K of vertices of T used to remember the set of branching nodes and parents of leaves; and
- labeling functions:
  - $\ell : V(\mathrm{cp}(T)) \cup E(T) \to \mathbb{N}$ used to remember the width of the partitioning tree;
  - $out : V(\mathrm{cp}(T)) \to \mathbb{N}$ used to remember the number of branching nodes that were "forgotten";
  - $branch : V(\mathrm{cp}(T)) \to \{0,1\}$ used to remember if the node in question is a branching node;
  - $father : V(\mathrm{cp}(T)) \to \{0,1\}$ used to remember if the node in question is not yet a branching node, but can become one if another child is added to it.

$\mathrm{Char}((T', r', \sigma'), B)$ is computed as follows.
1. Let T be the smallest subtree spanning the leaves of T' that map elements of B. Let r be the vertex of T that is closest to r' in T'. From now on, T is rooted in r. For any leaf f of T, let $\sigma(f) = \sigma'(f)$.
2. Let *dist* be the number of branching nodes on the path between r' and r in $\mathrm{cp}(T') \setminus r$. If $r \neq r'$, then $dist = 0$.
3. Let K be the set of vertices of T that are either a leaf of T, or the parent of a leaf of T, or a branching node of $\mathrm{cp}(T')$ in $V(T)$ (rooted in r'), or a branching node of (T, r). The last condition seems redundant, but is necessary in case the root of the tree changes, that is, in the case that $r' \neq r$.
4. For any vertex v of $V(\mathrm{cp}(T))$, $branch(v) = 1$ if v is a branching node of $\mathrm{cp}(T')$, and $branch(v) = 0$ otherwise.

For all $v \in V(T)$, let P_v be the set of paths between v and a leaf in $T' \setminus T$ all internal vertices of which are different from r and in $T' \setminus T$.

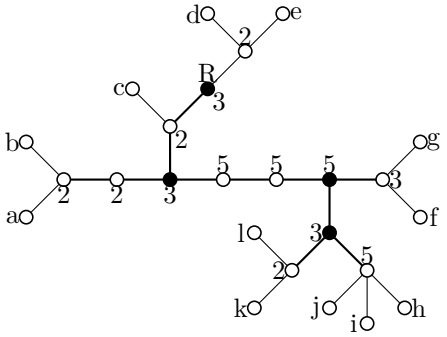
Let $out(v)$ be the maximum number of branching nodes that are internal to a path in P_v.

Let $father(v) = 1$ if v has a child that is not a leaf in T', otherwise let $father(v) = 0$.
5. Any internal vertex $v \in V(T)$ (resp., any edge $e \in E(T)$): $\ell(v) = \Phi_A(\mathcal{T}_v)$ (resp., $\ell(e) = \Phi_A(\mathcal{T}_e)$). Where $\mathcal{T}_v$ ($\mathcal{T}_e$) denotes the partition of A defined by v (e) in T'.
6. Then, in T, for any two vertices v, w in K such that no internal vertices of the path P between v and w are in K, replace P by $\mathrm{Contr}(P)$.

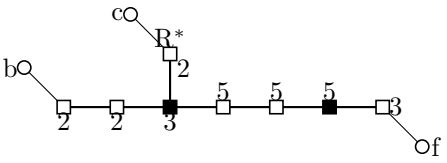
Remark 2. If $q = \infty$, we don't need to take the variables *dist*, *out*, *branch* and *father* into account. More precisely, the items 2, and 4 of the procedure Char can be removed and K is the set of vertices of T that are either a leaf of T, the parent of a leaf of T, or a branching node of (T, r).

We denote by $C = \mathrm{Char}((T, r, \sigma), B)$ the restriction of (T, r, σ) to B. Figure 5 illustrates the process Char when applied to a labeled partitioning tree of a set A.

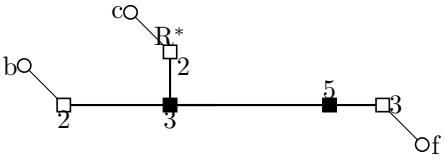
The key point for the understanding of the relationship between the partitioning tree (T', r', σ') of A and its restriction $((T, r, \sigma), \ell, K, dist, out, branch, father)$



(a) labeled partitioning tree



(b) before step 6



(c) characteristic after contraction of paths, labels other than ℓ are omitted

Figure 5: Partitioning tree (T, R, σ) of $\{a, b, \ldots, l\}$ and an execution of $\mathrm{Char}((T, R, \sigma), B)$ where $B = \{b, c, f\}$. Black nodes represent the branching nodes of T.

to B is based on the following. Any vertex of K represents a specific vertex of T' that is either a leaf of T' that maps an element of B, or the parent of such a leaf in T', or a branching node of $\mathrm{cp}(T')$ or a vertex of T' that defines a partition of B with at least three parts. Any path P between two vertices v, w in K such that no internal vertices of P between v and w are in K, represents a path $P(v, w)$ in T' the internal vertices of which have degree two in T'. Moreover, by definition of the operation $P = \mathrm{Contr}(P(v, w))$, any vertex (resp., edge) of P represents a specific vertex (resp., edge) of T'. Beside, by Lemma 15, the maximum (minimum) label over the vertices and edges of T is the maximum (minimum) label over the vertices and edges of T'. In particular, if (T', r', σ') has Φ-width at most k, then $\ell(v) \leq k$ and $\ell(e) \leq k$ for any $v \in V(T)$ and $e \in E(T)$.

Let the *br-height* of v, denoted by $\mathrm{brheight}_T(v)$, in $\mathrm{cp}(T)$ be the maximum number of branching nodes in a path from v to a leaf of the subtree of $\mathrm{cp}(T')$ rooted in v, i.e. v union the component of $T' \setminus v$ that does not contain r'. That is, $\mathrm{brheight}_T(v) = 0$ if v is a leaf of T, otherwise $\mathrm{brheight}_T(v) = \max\{out(v), \max_{u \text{ child of } v}\{\mathrm{brheight}(u)\}\} + branch(v)$. Lemma 19 shows that if a partitioning tree is q-branched then the brheight of the root of the restriction

of this partitioning tree is not bigger than q.

Lemma 19. *If (T', r', σ') is a q-branched partitioning tree for A, then* $\mathrm{Char}((T', r', \sigma'), B) = ((T, r, \sigma), \ell, K, dist, out, branch, father)$ *is such that* $\mathrm{brheight}_T(r) + dist \leq q$

Proof. Let T'' be the subtree of T' obtained by taking the union of all paths P in T' such that one extremity of P is r' and P passes through r. Clearly T (before Step 6) is a subtree of T''. Moreover, let $r'' = r'$ and σ'' be the restriction of σ' over the leaves of T''.

Then, the labels *dist*, *out* and *branch* are sufficient to remember if (T'', r'', σ'') is q-branched. By the definition of br-height, (T'', r'', σ'') is q-branched if and only if the br-height of r'' is at most q.

If v is a leaf of T, it is a leaf of T'', then $\mathrm{brheight}_T(v) = 0$. Otherwise, the br-height of $v \in V(T)$ is given by $\max\{out(v), height\} + branch(v)$, where *height* is the maximum of the br-height among the children of v.

In particular, if (T'', r'', σ'') is q branched, $out(v) \leq q$ for any $v \in K$. Finally, the $\mathrm{brheight}_T(r'') = \mathrm{brheight}_T(r) + dist \leq q$, since (T'', r'', σ'') is q branched. □

5.2.2 Characteristic of A restricted to B

Let $((T, r, \sigma), \ell, K, dist, out, branch, father)$ be such that (T, r, σ) is a rooted partitioning tree of $B \subseteq A$, $\ell : V(\mathrm{cp}(T)) \cup E(T) \to \mathbb{N}$, $K \subseteq V(T)$ that contains at least all leaves, parents of leaves, the root and vertices with degree at least three of T, $dist \in \mathbb{N}$, $out : V(\mathrm{cp}(T)) \to \mathbb{N}$, $branch : V(\mathrm{cp}(T)) \to \{0, 1\}$, $father : V(\mathrm{cp}(T)) \to \{0, 1\}$ and for any $v, w \in K$ such that no internal vertices of the path P between v and w are in K, $P = \mathrm{Contr}(Q)$ (i.e. P results from some contraction).

Definition 19. $((T, r, \sigma), \ell, K, dist, out, branch, father)$ is a *characteristic of A restricted to B* if it exists a partitioning tree (T', r', σ') of A, such that $((T, r, \sigma), \ell, K, dist, out, branch, father) = \mathrm{Char}((T', r', \sigma'), B)$. $((T, r, \sigma), \ell, K, dist, out, branch, father)$ is a *(k, q)-characteristic* of A restricted to B if, moreover, $\ell : V(\mathrm{cp}(T)) \cup E(T) \to [0, k]$, and $dist + \mathrm{brheight}_T(r) \leq q$.

Lemma 20. *If it exists a q-branched partitioning tree (T', r', σ') of A with Φ-width at most k, such that $C = \mathrm{Char}((T', r', \sigma'), B)$, then $C = ((T, r, \sigma), \ell, K, dist, out, branch, father)$ is a (k, q)-characteristic of A restricted to B*

Proof. This is a direct consequence of Definition 19 and Lemma 19. □

Definition 20. The size of a (k, q)-characteristic of A restricted to B, $((T, r, \sigma), \ell, K, dist, out, branch, father)$, is given by the expression $|V(T)| + |K|$.

Lemma 21. *If $q < \infty$, then the number of (k, q)-characteristic of A restricted to B, with $|B| = b$, is bounded by a function $f(k, q, b) = O((60kqb)^{45kqb})$, otherwise the number of (k, ∞)-characteristic of A restricted to B is bounded by a function $f(k, b) = O((15kb)^{45kb})$.*

Moreover, if $q < \infty$, then the size of a (k, q)-characteristic of A restricted to B is bounded by a function $f'(k, q, b) = O(kqb)$, otherwise the size of a (k, ∞)-characteristic of A restricted to B is bounded by a function $f'(k, b) = O(kb)$.

Proof. Let $((T, r, \sigma), \ell, K, dist, out, branch, father)$ be a (k, q)-characteristic of A restricted to B. T is a tree with b leaves.

If $q < \infty$, i.e., q is bounded.

Since $(T, r\sigma)$ is q-branched, for each leaf there are at most q branching nodes between r and this leaf. Therefore, $|K| \leq bq + 2b + 1$.

Any path between two vertices in K (that does not contain any other vertex in K) has at most $2k + 2$ internal vertices (Lemma 15). Hence, among all these paths there are at most $(bq + 2b)(2k + 2)$ vertices.

Let $n = (bq+2b)(2k+3)+1 \leq 15kqb$. The number of vertices in T is obtained by taking all vertices in K and all vertices that lies on a path between two vertices of K (that does not contain any other vertex in K). Thus, $|V(T)| \leq n$.

We can bound the maximum number of non-isomorph trees on n vertices by n^{n-2} [Cay89]. Let $\mathrm{T}(n) = \sum_{i=1}^{n} n^{n-2}$. The value $\mathrm{T}(n)$ is the number of different trees that can be the "base" of the characteristic. There are at most n vertices that can be the root of the tree and at most $b!$ (factorial) ways of mapping leaves of T to elements of B. Moreover, for each of these trees we can assign values for all the other variables, i.e., *branch*, *out*, *dist*, *father*, ℓ.

We have that $dist \leq q$ and for any vertex $v \in V(\mathrm{cp}(T))$ and edge $e \in E(T)$, $\ell(v) \leq k$, $\ell(e) \leq k$, $branch(v) \leq 1$, $out(v) \leq q$ and $father(v) \leq 1$.

Then, the number of different characteristics is bounded by:

$$\mathrm{T}(n)n(b!)q2^n q^n 2^n k^{2n-1} = 4^n q^{n+1} k^{2n-1}(b!)n\mathrm{T}(n).$$

Hence, the number of (k, q)-characteristic of A restricted to B is given by the function

$$f(k, q, |B|) = 4^n q^{n+1} k^{2n-1}(b!)n\mathrm{T}(n).$$

Since $n = O(15kqb)$, $q^{n+1} = O(q^{2n})$, $b! = O(b^{2n})$ and $\mathrm{T}(n) = O(n^n)$ with a coarse analysis we have that $f(k, q, b) = O((60kqb)^{45kqb})$.

To measure the size of one (k, q)-characteristic of A restricted to B, we can use the function $f'(k, q, b) \leq 2n$, since $|K| \leq n$.

If $q = \infty$, then $|K| \leq 3b$. Let $n' = 3b(2k + 3) < 15kb$ and $\mathrm{T}'(n') = \sum_{i=1}^{n} n'^{n'-2}$. Using the same reasoning as for the proof of the case $q < \infty$, we have that the number of (k, ∞)-characteristic of A restricted to B is given by a function:

$$f(k, b) = k^{2n'-1}(b!)\mathrm{T}'(n').$$

Since $b! = O(b^b) = O(b^{kb})$, $n' < 15kb$ and $\mathrm{T}'(n') = O(n'^{n'})$ with a coarse analysis we have that $f(k, b) = O((15kb)^{45kb})$

To measure the size of one (k, q)-characteristic of A restricted to B, we can use the function $f'(k, b) \leq n' + 3b$, since $|K| \leq 3b$. □

Definition 21. A set F of (k, q)-characteristics of A restricted to B is *full* if for all q-branched partitioning tree (T, r, σ) of A with Φ-width at most k, then $\mathrm{Char}((T, r, \sigma), B) \in F$.

6 Algorithm Using Characteristic

This section is devoted to the presentation of Procedures used in the decision algorithm for the q-branched Φ-width of a set A. Notations are those defined in Sections 2, 3 for Theorem 4.

Let $(D, \mathcal{X})$ be a nice decomposition for A that is compatible with a monotone partition function Φ, such that $\max_{t \in V(D)} |X_t| \leq k'$. Recall that for any $v \in V(D)$, D_v denotes the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

This section presents procedures that compute a full set $\mathrm{FSC}_{\mathrm{k,q}}(t)$ of (k,q)-charac-teristics of A_t restricted to X_t, for any $t \in V(T)$. The algorithm proceeds by dynamic programming from the leaves of D to its root.

Each procedure presented in this section takes as input a node $v \in V(D)$ and, for each u that is a child of v, the sets $\mathrm{FSC}_{\mathrm{k,q}}(u)$. The output of each procedure is $\mathrm{FSC}_{\mathrm{k,q}}(v)$.

6.1 Procedure *StartNode*

If v is a leaf, i.e. a start node of D, $A_v = X_v$, and $|X_v| \leq k'$. $\mathrm{FSC}_{\mathrm{k,q}}(v)$ consists of all (k,q)-characteristics of X_v.

Procedure *StartNode* enumerates all (k,q)-characteristics of A_v restricted to X_v.

Trivially, the following statement holds:

Lemma 22. *Procedure* StartNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v.*

Next lemma shows that the complexity of procedure *StartNode* does not depend on $|A|$.

Theorem 23. *Procedure* StartNode *has constant time complexity. That is, if $q < \infty$ then procedure* StartNode *computes* $\mathrm{FSC}_{\mathrm{k,q}}(v)$ *in time* $O((60kqk')^{46kqk'})$, *otherwise procedure* StartNode *computes* $\mathrm{FSC}_{\mathrm{k,q}}(v)$ *in time* $O((15kk')^{46kqk'})$.

Proof. By Lemma 21, $|\mathrm{FSC}_{\mathrm{k,q}}(v)|$ is bounded by the function $f(k,q,k')$, if $q < \infty$, or by the function $f(k,k')$, if $q = \infty$. Moreover, each element of $\mathrm{FSC}_{\mathrm{k,q}}(v)$ has a size bounded by the function by the function $f'(k,q,k')$, if $q < \infty$, or by the function $f'(k,k')$, if $q = \infty$.

Therefore, the amount of memory needed to store $\mathrm{FSC}_{\mathrm{k,q}}(v)$ has size bounded by $f(k,q,k') \cdot f'(k,q,k')$ in the case that $q < \infty$ or $f(k,k') \cdot f'(k,k')$ in the case that $q = \infty$.

Since $f(k,q,k') = O((60kqk')^{45kqk'})$ and $f'(k,q,k') = O(kqk')$ we have that the characteristics in $\mathrm{FSC}_{\mathrm{k,q}}(v)$ can be enumerated in $O((60kqk')^{46kqk'})$ in the case that $q < \infty$.

Since $f(k,k') = O((15kk')^{45kqk'})$ and $f'(k,k') = O(kk')$ we have that the characteristics in $\mathrm{FSC}_{\mathrm{k,q}}(v)$ can be enumerated in $O((15kk')^{46kk'})$ in the case that $q = \infty$. □

6.2 Procedure *IntroduceNode*

Let v be an introduce node of D, u its child, and $\{a\} = X_v \setminus X_u$. Let $\mathrm{FSC}_{\mathrm{k,q}}(u)$ be a full set of (k,q)-characteristics of A_u restricted to X_u. For each characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$, Procedure *IntroduceNode* proceeds as follows, repeating the five steps below, for any possible execution of Step 1. Roughly, it tries all possible ways to insert a into C_u obtaining C_v.

1. *update of T_u into T_v:*

   There are two ways of inserting a in C_u. Either choose an internal vertex v_{att} of $V(T_u)$, add a leaf v_{leaf} adjacent to v_{att} (*Case 1*), or choose an edge $f = \{v_{top}, v_{bottom}\}$ (with v_{top} closer to the root r_u than v_{bottom}), subdivide it into $e_{top} = \{v_{top}, v_{att}\}$ and $e_{bottom} = \{v_{att}, v_{bottom}\}$ and add a new leaf v_{leaf} adjacent to the new node v_{att} (*Case 2*). In both cases, σ_v keeps the same mapping as σ_u for leaves that are not v_{leaf}. We set $\sigma_v(v_{leaf}) = a$. Note that, now, T_v is a partitioning tree of X_v.

2. *update of labels of new vertex(ices) and edge(s):*

   In both cases of Step 1, $e_{new} = \{v_{leaf}, v_{att}\}$ receives label $\ell_v(e_{new}) \leftarrow \Phi_{A_v}(\{A_u, \{a\}\})$.

   In Case 2 of Step 1, v_{att} is a new vertex, then $\ell_v(v_{att}) = \ell_v(e_{top}) = \ell_v(e_{bottom}) \leftarrow \ell_u(f)$, and $out_v(v_{att}) = branch_v(v_{att}) \leftarrow 0$. If v_{bottom} is not a leaf of T_u, then $father_v(v_{att}) \leftarrow 1$. Otherwise, $father_v(v_{att}) \leftarrow 0$.

3. *update of labels of vertex(ices) and edge(s):*

   For each $e \in E(T_v)$, $e \neq e_{new}$, let $\mathcal{T}_e$ be the partition of X_v defined by e. $\ell_v(e) \leftarrow F_\Phi(\ell_u(e), \mathcal{T}_e, a)$.

   For each $t \in V(\mathrm{cp}(T_v))$, let $\mathcal{T}_t$ be the partition of X_v defined by t. $\ell_v(t) \leftarrow F_\Phi(\ell_u(t), \mathcal{T}_t, a)$.

   $dist_v \leftarrow dist_u$ and, for all internal vertex x of T_v, $out_v(x) \leftarrow out_u(x)$, $branch_v(x) \leftarrow branch_u(x)$ and $father_v(x) \leftarrow father_u(x)$.

   In Case 2 of Step 1, $father_v(v_{top}) \leftarrow 1$.

4. *creation of a new branching node:*

   In Case 1 of Step 1, $K_v \leftarrow K_u \cup \{v_{att}, v_{leaf}\}$.

   In Case 2 of Step 1, if v_{bottom} is the only child of v_{top} in T_u and $father_u(v_{top}) = 0$ (this implies that v_{top} belongs to K_u only because it is the parent of a single leaf), then $K_v \leftarrow (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$. Otherwise, $K_v \leftarrow K_u \cup \{v_{att}, v_{leaf}\}$.

   In Case 2 of Step 1, if v_{bottom} is a leaf of T_u and $father_u(v_{top}) = 1$, then

   $$branch_v(v_{top}) \leftarrow 1.$$

5. *contraction of paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

6. *update of* $\mathrm{FSC}_{\mathrm{k,q}}(v)$*:*

   $\mathrm{brheight}_T(r_v)$ is computable thanks to out_v and $branch_v$ as seen in Lemma 19. If $dist_v + \mathrm{brheight}_T(r_v) \leq q$ and $\ell_v(t) \leq k$ for any internal vertex $t \in V(T_v)$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, then $\mathrm{FSC}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSC}_{\mathrm{k,q}}(v) \cup \{C_v\}$.

The rest of this section is dedicated to show that procedure *IntroduceNode* computes $\mathrm{FSC}_{\mathrm{k,q}}(v)$ in constant time. We start by showing that the Procedure *IntroduceNode* computes a full set of (k, q)-characteristics of A_v restricted to X_v, then we analyse its complexity.

Lemma 24. *Procedure* IntroduceNode *computes a full set of* (k, q)*-characteristics of* A_v *restricted to* X_v*.*

Since Φ is closed under taking subset, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u does. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$. The proof of Lemma 24 is twofold. We first prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ returned by Procedure *IntroduceNode* is a set of characteristics of A_v restricted to X_v in Lemma 25, then we prove it is full in Lemma 26.

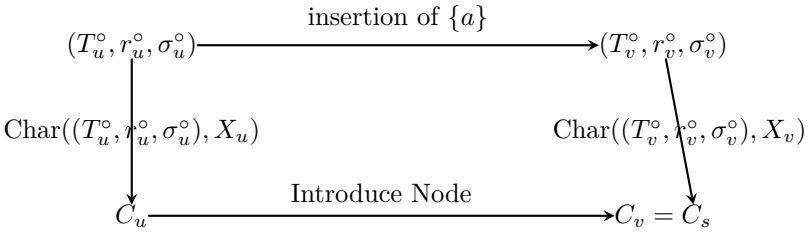


Figure 6: Scheme of proof of Lemma 24.

To prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a set of characteristics, we start from a partitioning tree $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ of A_u and its corresponding characteristic C_u. The insertion of a into C_u, from the *IntroduceNode*, results in C_v. By inserting a into $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ mimicking the insertion of a into C_u we obtain $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$. Then, we show that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. A scheme can be found in Figure 6.

Lemma 25. *For all* $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$*, we have that* C_v *is* (k, q)*-characteristic of* A_v *restricted to* X_v*.*

Proof. We introduce some notation in order to prove the lemma.

Let $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathit{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ be a characteristic used by procedure *IntroduceNode* to obtain C_v. That is, $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathit{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v) \in \mathrm{FSC}_{\mathrm{k,q}}(v)$ is an element of $\mathrm{FSC}_{\mathrm{k,q}}(v)$ constructed by applying the six steps of procedure *IntroduceNode* on $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$.

We assume that C_v is obtained from C_u with Case 2 of Step "update T_u into T_v" of Procedure *IntroduceNode*. That is, C_v is obtained from C_u by adding v_{leaf} as a neighbor of v_{att}, where v_{att} result from the subdivision of $f = \{v_{top}, v_{bottom}\} \in E(T_u)$. Case 1 of Step 1 can be proved in a similar way, thus we omit the proof here.

By definition $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$, hence it is a characteristic of a partitioning tree $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ of A_u restricted to X_u, i.e. $C_u = \mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$. Since $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is a full set of (k, q)-characteristics, we have that $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ is a q-branched partitioning tree for A_u with Φ-width at most k.

Note that, by definition of $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ and the "Contr" operation, f represents an edge $f_u^\circ \in E(T_u^\circ)$.

Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by subdividing the edge $f_u^\circ = \{v_{top}^\circ, v_{bottom}^\circ\}$ one time, creating a vertex v_{att}, and adding v_{leaf} as neighbor of v_{att}, make $\sigma_v^\circ(v_{leaf}) = a$ and $r_v^\circ = r_u^\circ$. By definition, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a partitioning tree of A_v.

Let $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, dist_s, out_s, branch_s, father_s) = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, i.e. C_s is the restriction of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ to X_v.

We need to show that $C_v = C_s$.

In order to prove that $C_v = C_s$, we need to introduce some notation.

For all $t \in V(\text{cp}(T_v^\circ))$, let $\mathcal{A}_t^\circ$ be the partition of A_v defined by t. Similarly, for all $e \in E(T_v^\circ)$, let $\mathcal{A}_e^\circ$ be the partition of A_v defined by e.

Let T_v' be the smallest subtree of T_v° the leaves of which map all elements of X_v, and let r_v' be the vertex in T_v' that is closest to r_v°. Similarly, let T_u' be the smallest subtree of T_u° the leaves of which map all elements of X_u, and let r_u' be the vertex in T_u' that is closest to r_u°. Note that, T_u' and T_v' are obtained in the first step of the procedure Char when applied to $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ and $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ respectively.

For the remainder of this section, assume that $V(T_v^\circ) = \{1, \ldots, n\}$, where $v_{leaf} = n$ and $v_{att} = n-1$. Then, from the definition of T_v°, $V(T_u^\circ) = V(T_v^\circ) \setminus \{n-1, n\}$. Moreover, from the Char procedure, $V(T_s) \subseteq V(T_v^\circ)$ and $V(T_u) \subseteq V(T_u^\circ)$ and, from the *IntroduceNode* procedure $V(T_v) \subseteq V(T_u) \cup \{v_{leaf}, v_{att}\}$.

Claim 2. *The sets K_v and K_s are the same, i.e. $K_v = K_s$.*

By step 3 of the procedure Char, K_s is the set of vertices that are leaves in (T_v', r_v'), parents of leaves, branching nodes of $\text{cp}(T_v^\circ)$ in $V(T_v')$, or branching nodes of (T_v', r_v'). T_v° can be obtained from T_u° by subdividing f_u° and adding a new leaf v_{leaf} adjacent to the new vertex v_{att} and $\sigma_v^\circ(v_{leaf}) = a$, therefore T_v' is obtained from T_u' by subdividing f_u° and adding a neighbor to the vertex created from the subdivision.

$K_s \setminus \{v_{top}\}$ is composed by vertices that are leaves in (T_u', r_u'), or parents of leaves, or branching nodes of $\text{cp}(T_u^\circ)$ in $V(T_u')$, or branching nodes of (T_u', r_u'), or v_{att}, or v_{leaf}. In other words, $K_s \setminus \{v_{top}\} = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$.

There are 2 cases to consider: (1) v_{bottom} is the unique child of v_{top} in T_u and $father_u(v_{top}) = 0$; or (2) otherwise.

Case (1): $K_v = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$. Since v_{bottom} is the unique child of v_{top} in T_u and $father_u(v_{top}) = 0$, v_{top} has only one child in T_u' and it is not a branching node of T_u°. From the construction of T_v°, the only child of v_{top} in T_v' is v_{att} which is not a leaf and v_{top} is not a branching node of T_v°. Hence, $v_{top} \notin K_s$. Therefore, $K_s \setminus \{v_{top}\} = K_s = K_v$.

Case (2): either v_{top} has more than one child in T_u or $father_u(v_{top}) = 1$. Then, $K_v = K_u \cup \{v_{att}, v_{leaf}\}$, from step "creation of a new branching node" of *IntroduceNode*. There are some sub-cases to consider:

- If v_{top} has more than one child in T_u, then v_{top} is a branching node of (T_u', r_u'), hence $v_{top} \in K_u$. From the construction of T_v', v_{top} is a branching node of (T_v', r_v'), therefore $v_{top} \in K_s$. Then, since $v_{top} \in K_u$ and $K_s \setminus \{v_{top}\} = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$ we have that $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.
- If $father_u(v_{top}) = 1$ and v_{bottom} is a leaf of T_u, then $v_{top}^\circ \in V(T_u^\circ)$ is the parent of a leaf of T_u°, hence $v_{top} \in K_u$. From the construction of T_v°, v_{top} is a branching node of $\text{cp}(T_v^\circ)$, hence $v_{top} \in K_s$. Then, $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.

- If $father_u(v_{top}) = 1$ and v_{bottom} is not a leaf of T_u. From the construction of T_v°, v_{top} is a branching node of T_v° if and only if v_{top} is a branching node of T_u°. Hence, $v_{top} \in K_u$ if and only if $v_{top} \in K_s$. Therefore, $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.

In both cases, we have $K_v = K_s$.

Claim 3. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

Recall that, from the Char procedure, T_s is obtained by contracting all paths of T_v° that have endpoints in K_s and no internal vertex in K_s. On the other hand, T_v is obtained by adding a v_{leaf} adjacent to v_{att} in T_u and then contracting all paths of T_v that have endpoints in K_v and no internal vertex in K_v. We want to show that the result of the contractions in T_v° and the contractions in T_v are the same. That is, that after contractions T_v and T_s are isomorph having the same labels on its vertices.

Let $P_v(x, y)$ denote the labeled path between vertices x and y in T_v with labels ℓ_v (resp. $P_s(x, y)$ in T_s with labels ℓ_s). In order to show these two properties, we show that for each pair of vertices $x, y \in K_v = K_s$ (Claim 2) such that the path $P_v(x, y)$ has no vertex from K_v as internal vertex then $P_v(x, y) = P_s(x, y)$. In other words, the paths in T_v and T_s between two vertices of $K_v = K_s$ have the same length and have the same sequence of labels defined by the functions ℓ_v and ℓ_s respectively.

Let $P'_s(x, y)$ be the path between x and y in T'_v. In other words, $P'_s(x, y)$ is the path $P_s(x, y)$ of C_s along with labels given by ℓ_s before Step 6 of Char$((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

There are some cases to consider: (1) $P_s(x, y) = (v_{att}, v_{leaf})$; (2) $P_s(x, y) \neq (v_{att}, v_{leaf})$ and $v_{att} \notin \{x, y\}$; (3) $P_s(x, y) \neq (v_{att}, v_{leaf})$ and $v_{att} \in \{x, y\}$.

Case (1): If $P_s(x, y) = \{v_{att}, v_{leaf}\}$, by Step "update labels of new vertex(s) and edge(s)" of Procedure *IntroduceNode* $\ell_v(\{v_{att}, v_{leaf}\}) = \ell_s(\{v_{att}, v_{leaf}\})$. From the fact that C_s is obtained through the Char procedure, we have that $\ell_s(v_{att}) = \Phi_{A_v}(\mathcal{A}_v^\circ)$. From the fact that Φ is compatible with $(D, \mathcal{X})$, we have that $\Phi_{A_v}(\mathcal{A}_v^\circ) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a)$. Finally, from the step "update of labels of vertices and edges" of procedure *IntroduceNode* we have that $\ell_v(v_{att}) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a)$. Taking all these inequalities we have that $\ell_s(v_{att}) = \Phi_{A_v}(\mathcal{A}_v^\circ) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a) = \ell_v(v_{att})$.

Case (2): Now, let us assume that $P_s(x, y) \neq \{v_{att}, v_{leaf}\}$ and $v_{att} \notin \{x, y\}$. Then, $P_s(x, y)$ represents a path P_u° in T_u°, and more precisely in T'_u. Each $t \in V(P_u^\circ)$ defines a partition $\mathcal{T}_t^\circ$ of A_u such that $\Phi_{A_u}(\mathcal{T}_t^\circ) = \Phi_{A_v}(\mathcal{A}_t^\circ) \cap A_u$. Similarly, each $e \in E(P_u^\circ)$ defines a partition $\mathcal{T}_e^\circ$ of A_u such that $\Phi_{A_u}(\mathcal{T}_e^\circ) = \Phi_{A_v}(\mathcal{A}_e^\circ) \cap A_u$. Moreover, the labels in P_u° are given by the function Φ_{A_u} applied to the partitions of A_u defined by the vertices (or edges) of P_u°.

When computing Char$((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ to obtain C_u, P_u° is replaced by Contr(P_u°). Each internal vertex and edge of Contr(P_u°) defines the same partition $\mathcal{P}$ of X_v (where a is in the part correspondent to the component where edge f is), since these vertices are not in K_u, hence they are not

leaves, nor parent of leaves of X_u, nor branching nodes of $\mathrm{cp}(T_u^\circ)$ and f_u° does not belong to P_u°.

To obtain $P_v(x,y)$, procedure *IntroduceNode* modifies the labels of edges and vertices of $\mathrm{Contr}(P_u^\circ)$ by applying the strictly increasing function $F_{\Phi,\mathcal{P}} : x \to F_\Phi(x,\mathcal{P},a)$ (Step "update of labels of vertex(s) and edge(s)" of Procedure *IntroduceNode*), then, let $F_{\Phi,\mathcal{P}}(\mathrm{Contr}(P_u^\circ))$ be the path obtained in this way, then it replaces $F_{\Phi,\mathcal{P}}(\mathrm{Contr}(P_u^\circ))$ by $\mathrm{Contr}(F_{\Phi,\mathcal{P}}(\mathrm{Contr}(P_u^\circ)))$. Hence, $\mathrm{Contr}(F_{\Phi,\mathcal{P}}(\mathrm{Contr}(P_u^\circ))) = P_v(x,y)$. By Items 2 and 1 of Lemma 16,

$$P_v(x,y) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(\mathrm{Contr}(P_u^\circ))) = \mathrm{Contr}(\mathrm{Contr}(F_{\Phi,\mathcal{P}}(P_u^\circ))) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P_u^\circ)).$$

From the definition of T_v° and the fact that Φ is compatible with $(D,\mathcal{X})$, we have that $F_{\Phi,\mathcal{P}}(P_u^\circ) = P_v^\circ$, hence, by Step 6 of $\mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, $\mathrm{Contr}(F_{\Phi,\mathcal{P}}(P_u^\circ)) = P_s(x,y)$. Therefore, $P_v(x,y) = P_s(x,y)$.

Case (3): Let x and y be the vertices in K_u such that the path P_u° of T_u° between x and y contains the edge f_u°. It remains to prove that $P_v(x,v_{att}) = P_s(x,v_{att})$ and $P_v(v_{att},y) = P_s(v_{att},y)$.

Let C_u' be obtained from C_u by subdividing f resulting in new vertex v_{att}. Moreover, let $\ell_u'(v_{att}) = \ell_u'(\{v_{top}, v_{att}\}) = \ell_u'(\{v_{att}, v_{bottom}\}) = \ell_u(f)$. From the procedure *IntroduceNode* $P_v(x,v_{att})$ is obtained from $P_u'(x,v_{att})$ by applying the function F and then a contraction and $P_v(v_{att},y)$ is obtained from $P_u'(v_{att},y)$ by applying the function F and then a contraction. That is, $P_v(x,v_{att}) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P_u'(x,v_{att})))$, where $\mathcal{P}$ is the partition of $\mathcal{P}$ is the partition of X_v defined by vertices and edges in $P_v(x,v_{att})$. Similarly, $P_v(v_{att},y) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P_u'(v_{att},y)))$, where $\mathcal{P}'$ is the partition of X_v defined by vertices and edges in $P_v(v_{att},y)$.

On the other hand, $P_s(x,v_{att})$ is obtained by applying a contraction on the path from x to v_{att} in T_v° and $P_s(v_{att},y)$ is obtained by applying a contraction on the path from v_{att} to y.

Let T_u'' be the tree obtained from T_u° by subdividing f_u° resulting in new vertex v_{att}, i.e. T_u'' is the tree T_v° without v_{leaf}. Let $P_u''(x,y) = P_u''(x,v_{att}) \odot P_u''(v_{att},y)$ be the path in T_u'' between x and y. Then, $P_v^\circ(x,y) = P_v^\circ(x,v_{att}) \odot P_v^\circ(v_{att},y)$ is the path in T_v° corresponding to $P_u''(x,y)$ in T_u''. That is, $P_v^\circ(x,y)$ has the same vertices as $P_u''(x,y)$ but with different labels.

Note that the partition of X_v defined by the vertices and edges in $P_v^\circ(x,v_{att})$ is the same as the one defined by $P_v(x,v_{att})$, that is, $\mathcal{P}$. Similarly, the partition of X_v defined by the vertices and edges in $P_v^\circ(v_{att},y)$ is the same as the one defined by $P_v(v_{att},y)$, that is, $\mathcal{P}'$. From the fact that the fact Φ is compatible with $(D,\mathcal{X})$, we have that $P_v^\circ(x,v_{att}) = F_{\Phi,\mathcal{P}}(P_u''(x,v_{att}))$ and that $P_v^\circ(v_{att},y) = F_{\Phi,\mathcal{P}'}(P_u''(v_{att},y))$. Therefore, $P_s(x,v_{att}) = \mathrm{Contr}(P_v^\circ(x,v_{att})) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P_u''(x,v_{att})))$ and $P_s(v_{att},y) = \mathrm{Contr}(P_v^\circ(v_{att},y)) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P_u''(v_{att},y)))$.

Then, taking all these inequalities, we have that:

$$P_v(x, v_{att}) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P'_u(x, v_{att}))),$$
$$P_v(v_{att}, y) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P'_u(v_{att}, y))),$$
$$P_s(x, v_{att}) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P''_u(x, v_{att}))) \text{ and}$$
$$P_s(v_{att}, y) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y))).$$

Note that $P'_u(x, y) = \mathrm{Contr}(P''_u(x, y))$, then by Lemma 17:

$$P_s(x, v_{att}) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P''_u(x, v_{att}))) = \mathrm{Contr}(F_{\Phi,\mathcal{P}}(P'_u(x, v_{att}))) = P_v(x, v_{att}), \text{ and}$$
$$P_s(v_{att}, y) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y))) = \mathrm{Contr}(F_{\Phi,\mathcal{P}'}(P'_u(v_{att}, y))) = P_v(v_{att}, y).$$

Since $K_v = K_s$ (Claim 2, and, in all cases, for each x, $y \in K_s$, $P_v(x, y) = P_s(x, y)$, we have that T_v is isomorphic to T_s and that ℓ_v and ℓ_s are equivalent, that is, correspondent vertices and edges in T_v and T_s have the same label.

Claim 4. $(\mathit{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v) = (\mathit{dist}_s, \mathit{out}_s, \mathit{branch}_s, \mathit{father}_s)$.

By procedure *IntroduceNode*, dist_v receives the value of dist_u, i.e. the number of branching nodes in T_u° between r_u° and r'_u. We have that dist_u is the number of branching nodes in T_v° between r_v° and r'_s, i.e. dist_s. Hence, $\mathit{dist}_v = \mathit{dist}_s$.

Now, for every vertex t in $\mathrm{cp}(T_v)$, $\mathit{out}_v(t)$ is the maximum number of branching nodes on a path between t and a leaf in $A_u \setminus X_u$ every internal vertices of which are different from r_u° and in $T'_u \setminus T_u^\circ$. It is also the maximum number of branching nodes on a path between t and a leaf in $A_v \setminus X_v = A_u \setminus X_u$ every internal vertices of which are different from r_v° and in $T'_s \setminus T_v^\circ$, i.e. $\mathit{out}_s(t)$.

For every vertex t in $\mathrm{cp}(T_v)$, $\mathit{branch}_v(t) = 1$ if and only if $\mathit{branch}_u(t) = 1$ or t is the parent-end of f and $\mathit{father}_u(t) = 1$ (Step "creation of a new branching node" of Procedure *IntroduceNode*). That is, $\mathit{branch}_v(t) = 1$ if and only if t is a branching node of $\mathrm{cp}(T_v^\circ)$, i.e. $\mathit{branch}_v(t) = \mathit{branch}_s(t)$.

Now, for every vertex t in $\mathrm{cp}(T_v) \setminus \{v_{top}\}$, $\mathit{father}_v(t) = \mathit{father}_u(t)$ and $\mathit{father}_v(v_{top}) = 1$. In other words, $\mathit{father}_v(t) = 1$ if and only if t is the parent of a non leaf node in T_v°. Since, $\mathit{father}_u(t) = \mathit{father}_s(t)$ for every vertex $t \in \mathrm{cp}(T_u)$, $\mathit{father}_v(t) = \mathit{father}_s(t)$. Moreover, $\mathit{father}_s(v_{top}) = 1$.

Claim 5. *$(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a q-branched partitioning tree for A_v with Φ-width not bigger than k.*

Therefore, we proved that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. By Step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$" of Procedure *IntroduceNode*, we have that $\mathit{dist}_v + \mathrm{brheight}_T(r_v) \leq q$. Note that, since (T_u°, r_u°) is q-branched, for every path P in $\mathrm{cp}(T_v^\circ)$ from r_v° to a leaf of $\mathrm{cp}(T_v^\circ)$ such that P does not pass through r'_v we have that P has at most q branching nodes. Hence, by Lemma 19 $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched.

It remains to show that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ width at most k. Consider any internal vertex t of $V(T_v^\circ) \setminus V(T'_v)$. Let $\mathcal{P}_t$ be the partition of A_v defined by t. Since t is not in $V(T'_v)$ the partition of X_v it defines has only one part. That is, the partition $\mathcal{P}_t \cap X_v$ defined by t has at most one part. From the fact that Φ is compatible with $(D, \mathcal{X})$ we have that $\Phi_{A_v}(\mathcal{P}_t) = \Phi_{A_v}(\mathcal{P}_t) = \Phi_{A_u}(\mathcal{P}_t \cap A_u)$. Then, from the fact that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ width at most k we have that $\Phi_{A_v}(\mathcal{P}_t) \leq k$.

Similarly, for any edge e of $E(T_v^\circ) \setminus E(T'_v)$ we have that $\Phi_{A_v}(\mathcal{P}_e) \leq k$, where $\mathcal{P}_e$ is the partition of A_v defined by e.

From the definition of "Contr", item 1 from Lemma 15 and the fact that $\ell_v(t) \leq k$ for any vertex $t \in V(\text{cp}(T_v))$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, we have that (T'_v, r'_v, σ'_v) has Φ-width at most k. Hence, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ-width at most k.

Therefore, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a q-branched partitioning tree with Φ-width at most k. Thus, $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ is a (k,q)-characteristic of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ restricted to X_v.

This concludes the proof, that is $\text{FSC}_{\text{k,q}}(v)$ is a set of (k,q)-characteristics restricted to A_v. □

To prove that the set $\text{FSC}_{\text{k,q}}(v)$ is full, we consider an arbitrary q-branched partitioning tree $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ with Φ-width not bigger than k of A_v and show that there is an execution of *IntroduceNode* on a characteristic $C_u \in \text{FSC}_{\text{k,q}}(u)$ such that $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ and $C_v \in \text{FSC}_{\text{k,q}}(v)$.

Lemma 26. *The set* $\text{FSC}_{\text{k,q}}(v)$*, computed through the procedure* IntroduceNode*, is full.*

Proof. Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be a q-branched partitioning tree of A_v with Φ-width not bigger than k. Let v_{leaf} be the leaf of T_v° that maps $\{a\} = X_v \setminus X_u$. Let v_{att} be the parent of v_{leaf}. Let $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ be the partitioning tree of A_u such that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ can be obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by inserting a vertex correspondent to a as a neighbor of v_{att} in T_u°. Therefore, $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ is a q-branched partitioning tree for A_u with Φ-width not bigger than k. Let C_v be such that $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. It remains to show that $C_v \in \text{FSC}_{\text{k,q}}(v)$.

From the induction hypothesis, there is $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ in $\text{FSC}_{\text{k,q}}(u)$. Then, there are three cases to consider: (1) $v_{att} \in K_u$; (2) $v_{att} \notin K_u$ and v_{att} is represented in C_u; (3) $v_{att} \notin K_u$ and v_{att} is not represented in C_u.

Cases (1) and (2): Then, v_{att} is a vertex of T_u, i.e. during the Step 6 of $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ the vertex v_{att} does not suffer a contraction operation. Consider the execution of case 1 of procedure *IntroduceNode* on C_u, where v_{leaf} is added as a neighbor to v_{att}, to obtain C_v. In other words, T_v can be obtained from T_u by adding a vertex v_{leaf} as a neighbor of v_{att}. Hence, from the step "update of labels of vertices and edges" of *IntroduceNode* procedure and the fact that Φ is compatible with $(D, \mathcal{X})$, the labels ℓ_v of C_v obtained from the labels ℓ_u of C_u are such that for any internal vertex (or edge) t in T_v:

$$\ell_v(t) = F_\Phi(\ell_u(t), \mathcal{T}_t, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \mathcal{A}_t \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_t)$$

Where $\mathcal{T}_t$ and $\mathcal{A}_t$ are the partitions of X_v and A_v defined by t respectively. From the Step "update of labels of new vertex(s) and edge(s)" of *IntroduceNode* procedure $\ell_v(\{v_{att}, v_{leaf}\}) = \Phi_{A_v}(\{A_u, \{a\}\})$.

We need to show that in step "update of $\text{FSC}_{\text{k,q}}(v)$" we add C_v to $\text{FSC}_{\text{k,q}}(v)$. Since $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched and its Φ-width is not bigger than k and $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we have that $\text{brheight}(r_v) \leq q$ and that, for all internal vertices (or edges) t of T_v, $\ell_v(t) \leq k$ (Lemma 20). Hence, during step "update of $\text{FSC}_{\text{k,q}}(v)$" we have that C_v is inserted into $\text{FSC}_{\text{k,q}}(v)$.

Case (3): The vertex v_{att} does not belong to K_u nor has a representative on C_u. This means that v_{att} is contracted during the operation $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ))$. Let $P'_u(x,y)$ be the path between x and y in T'_u (the subtree of T_u° spanning leaves mapping elements of X_u) such that $v_{att} \in V(P'_u(x,y))$ and $x, y \in K_u$. In other words, in Step 6 of $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ to obtain C_u during the contraction of $P'_u(x,y)$ we have that v_{att} is removed with a contraction operation. Let $P_u(x,y) = \mathrm{Contr}(P'_u(x,y))$, i.e. the path in C_u resulting from the contraction of $P'_u(x,y)$. Thus, there are vertices x' and y' in $V(P_u(x,y))$ such that v_{att} is an internal node of $P'_u(x',y')$ and $\{x',y'\}$ is an edge of $P_u(x,y)$. In other words, v_{att} is removed either by a contraction between $\{x',y'\}$ and x' or by a contraction between $\{x',y'\}$ and y'.

We want to show that C_v, obtained through an execution of case 2 of *IntroduceNode* on C_u where the edge $\{x',y'\}$ is subdivided, is such that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ and $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$.

Consider the execution of procedure *IntroduceNode* on C_u by applying case 2 on the edge $\{x',y'\}$. In this execution of *IntroduceNode*, T_v is obtained from T_u by subdividing $\{x',y'\}$ creating a vertex v_{att} and adding v_{leaf}, mapping a, as a neighbor of v_{att}.

Using the same argument as in "proof of cases (1) and (2)" we have that for any internal vertex (or edge) t in T_v, $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$ where $\mathcal{A}_t$ is the partition of A_v defined by t.

We need to show that in step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$" we add C_v to $\mathrm{FSC}_{\mathrm{k,q}}(v)$. Since $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched and its Φ-width is not bigger than k and $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we have that $\mathrm{brheight}(r_v) \leq q$ and that, for all internal vertices (or edges) t of T_v, $\ell_v(t) \leq k$ (Lemma 20). Hence, during step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$" we have that C_v is inserted into $\mathrm{FSC}_{\mathrm{k,q}}(v)$.

Since, in all these cases, we have that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$, this shows that $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of (k,q)-characteristics for A_v restricted to X_v. □

Theorem 27. *Procedure* IntroduceNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity of procedure* IntroduceNode *is bounded by a a function $f_i(k,q,k') = O((60kqk')^{45kqk'}(kqk')^4)$, if $q < \infty$, or a function $f'_i(k,k') = O((15kk')^{45kk'}(kk')^4)$ otherwise.*

Proof. From Theorem 24, procedure *IntroduceNode* computes a full set of (k,q)-charac-teristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted. From its definition, F_Φ can be computed in constant time. Therefore, for each element $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ steps 1 to 5 can be done in $O(|T_u|)$, for each possible execution of Step 1.

Contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges and verifying if a contraction operation can be done between them. Hence, step 5 can be executed in $O(|T_u|^3)$, for each possible execution of Step 1.

Lastly, step 6 can be executed in $O(|T_u|)$, by traversing the tree T_u in a bottom up order, for each possible execution of Step 1.

Since the size and the number of elements in $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is bounded by $f'(k,q,k') = O(kqk')$ and $f(k,q,k') = O((60kqk')^{45kqk'})$ respectively, if $q < \infty$, or by $f'(k,k') = O(kk')$ and $f(k,k') = O((15kk')^{45kk'})$ if $q = \infty$ from Lemma 21, we get the result. That is, since there are at most $O(|T_u|)$ different executions for Step 1, the complexity of procedure *IntroduceNode* is bounded by, if $q < \infty$:

$$f_i(k,q,k') = O\left(f(k,q,k') \cdot f'(k,q,k')^4\right) = O((60kqk')^{45kqk'}(kqk')^4).$$

In the case that $q = \infty$ the complexity of procedure *IntroduceNode* is bounded by:

$$f_i(k,k') = O\left(f(k,k') \cdot f'(k,k')^4\right) = O((15kk')^{45kk'}(kk')^4).$$

□

6.3 Procedure *ForgetNode*

Let v be a forget node of D, u be its child and $\mathrm{FSC}_{\mathrm{k,q}}(u)$ be a full set of (k,q)-characteristics of A_u restricted to X_u. For every characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathit{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$, Procedure *ForgetNode* proceeds as follows. Roughly, it restricts C_u to $X_v = X_u \setminus \{a\}$ obtaining C_v.

1. *preparation:*

   Let v_{leaf} be the leaf of T_u that maps a, let v_{att} be the vertex of T_u with degree at least three that is closest to v_{leaf} (if no such a vertex exists, T_u is a path and v_{att} is set to the only other leaf of the path). Let P be the path between v_{leaf} and v_{att}. Let w be the neighbor of v_{att} in P. Let p be the number of vertices $y \in V(T_u) \setminus \{v_{\mathit{att}}\}$ with $\mathit{branch}_u(y) = 1$ in the path between v_{att} and r_u.

2. *removing a from T_u:*

   T_v is obtained by removing $V(P) \setminus \{v_{\mathit{att}}\}$ and $E(P)$ from T_u.

   If $r_u \neq v_{\mathit{att}}$ and r_u belongs to the path P between v_{leaf} and v_{att}:

   - $r_v \leftarrow v_{\mathit{att}}$,
   - $\mathit{dist}_v \leftarrow \mathit{dist}_u + p$, and
   - $\mathit{out}_v(v_{\mathit{att}}) \leftarrow \mathit{out}_u(v_{\mathit{att}})$.

   If $r_u = v_{\mathit{att}}$ or r_u does not belong to P:

   - $r_v \leftarrow r_u$,
   - $\mathit{dist}_v \leftarrow \mathit{dist}_u$, and
   - $\mathit{out}_v(v_{\mathit{att}}) \leftarrow \max\{\mathit{out}_u(v_{\mathit{att}}), \mathrm{brheight}_T(w)\}$.

   In either case:

   - $\mathit{branch}_v(v_{\mathit{att}}) \leftarrow \mathit{branch}_u(v_{\mathit{att}})$, and
   - $\mathit{father}_v(v_{\mathit{att}}) \leftarrow \mathit{father}_u(v_{\mathit{att}})$.

For every vertex $x \in \mathrm{cp}(T_v)$ such that $x \neq v_{att}$:

- $out_v(x) \leftarrow out_u(x)$,
- $branch_v(x) \leftarrow branch_u(x)$, and
- $father_v(x) \leftarrow father_u(x)$.

3. *updating K_v:*

   If v_{att} has degree two in T_v, and v_{att} is not the parent of a leaf neither the root in T_v, and $branch_u(v_{att}) = 0$, then $K_v \leftarrow K_u \setminus V(P)$.

   Otherwise, K_v is obtained by removing $V(P) \setminus \{v_{att}\}$ from K_u.

4. *contracting the paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

5. *updating* $\mathrm{FSC}_{\mathrm{k,q}}(v)$*:*

   Add C_v to $\mathrm{FSC}_{\mathrm{k,q}}(v)$.

The rest of this section is dedicated to proving Lemma 28.

Since $A_v = A_u$, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u does. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$. A scheme of the proof of Lemma 28 can be found in Figure 7

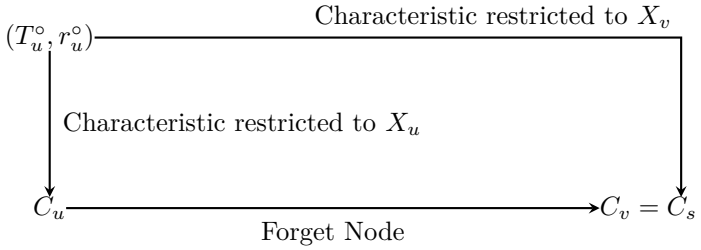


Figure 7: Scheme of proof of Lemma 28.

Lemma 28. ForgetNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v.*

Proof. Let $(T^\circ, r^\circ, \sigma^\circ)$ be any q-branched partitioning tree for A_u with Φ-width at most k. Since $A_v = A_u$, we have that $(T^\circ, r^\circ, \sigma^\circ)$ is a q-branched partitioning tree for A_u with Φ-width at most k if an only if it is also a q-branched partitioning tree for A_v with Φ-width at most k.

Since $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is a full set of (k,q)-characteristics for A_u restricted to X_u, we have that there exists $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ which is a (k,q)-characteristic of $(T^\circ, r^\circ, \sigma^\circ)$ restricted to X_u. In other words, $C_u = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_u)$. Let C_v be obtained through procedure *ForgetNode* when applied to C_u, by removing the path $P_u(v_{leaf}, v_{att})$ from T_u, where v_{leaf} is the leaf of T_u mapping a.

We want to show that $C_v = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$ and that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$. Since step "updating $\mathrm{FSC}_{\mathrm{k,q}}(v)$" from procedure, we have that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$. It remains to show that $C_v = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$.

In order to do that, let $C_s = \text{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$, we need to show that $C_s = C_v$. Let $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, dist_s, out_s, branch_s, father_s)$ and let $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, dist_v, out_v, branch_v, father_v)$.

To prove that $C_v = C_s$, we must introduce some notation. Let T'_u (resp. T'_v) be the minimum subtree of T° that contains all leaves that maps elements of X_u (resp. X_v).

From the Char procedure, we have that T_u is a path if and only if T'_u is a path. If T'_u is a path, then $|X_u| = 2$. This means that, $|X_v| = 1$ and, hence, T'_v is a single vertex. Then, from the procedure $\text{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$, we have that C_s is a characteristic of A_v restricted to X_v where T_s is a single vertex. On the other hand, from the step "preparation" of procedure *ForgetNode*, if T_u is a path, we have that T_v is also a single vertex. Hence, it is easy to see that when T'_v is a single vertex $C_v = C_s$.

Therefore, assume that T'_u is not a path and let $v_{att} \in T'_u$ be the vertex with degree at least three that is closest to v_{leaf}, the vertex mapping a.

For the remainder of this section, assume that $V(T^\circ) = \{1, \ldots, n\}$. From the Char procedure, $V(T_s) \subseteq V(T^\circ)$ and $V(T_u) \subseteq V(T^\circ)$ and, from the *ForgetNode* procedure $V(T_v) \subset V(T_u)$.

Claim 6. *The sets K_v and K_s are the same, i.e., $K_s = K_v$.*

Let T'_v be the minimum subtree of T° that contains all leaves that maps elements of X_v. Clearly, T'_v is a subtree of T'_u. Let r'_v be the vertex of T'_v that is closest to r°. From the definition of C_s, K_s is the set of vertices of T'_v that are either a leaf of T'_v, or the parent of a leaf of T'_v, or a branching node of $\text{cp}(T^\circ)$ in $V(T'_v)$ (rooted in r°), or a branching node of (T'_v, r'_v).

Since $C_u = \text{Char}((T^\circ, r^\circ, \sigma^\circ), X_u)$, K_u is the set of vertices that are either a leaf of T'_u, or the parent of a leaf of T'_u, or a branching node of $\text{cp}(T^\circ)$ in $V(T'_u)$ (rooted in r°), or a branching node of (T'_u, r'_u). Therefore, since T'_v is a subtree of T'_u and all leaves in T'_v are leaves in T'_u we have that $K_s \subseteq K_u$. Moreover, T'_v is the tree obtained from T'_u by removing the vertex v_{leaf} and all internal vertices of $P(v_{leaf}, v_{att})$, a path between v_{leaf} and v_{att} in T'_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}))$ or $K_s = (K_u \setminus V(P(v_{leaf}, v_{att})) \cup \{v_{att}\})$. There are two cases to consider:

Case (1): Assume that the vertex v_{att} does not belong to K_s. That is, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}))$. Then, v_{att} is not the parent of a leaf in T'_v, nor a branching node of $\text{cp}(T^\circ)$, nor a branching node of (T'_v, r'_v). Since, v_{att} is not the parent of a leaf in T'_v and it is not a branching node of (T'_v, r'_v) it has degree two in T'_v. Therefore, v_{att} is not the parent of a leaf, different than v_{leaf}, in T'_u, the variable $branch_u(v) = 0$, and v_{att} has degree two in T'_v. Consequently, by Step "updating K_v", $v_{att} \notin K_v$ and $K_v = K_u \setminus V(P_u(v_{leaf}, v_{att}))$, where $P_u(v_{leaf}, v_{att})$ is the path between v_{leaf} and v_{att} in T_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}) = K_u \setminus V(P_u(v_{leaf}, v_{att})) = K_v$.

Case (2): Assume that $v_{att} \in K_s$. That is, $K_s = (K_u \setminus V(P(v_{leaf}, v_{att})) \cup \{v_{att}\})$. If v_{att} belongs to K_s, then it is the parent of a leaf in T'_v, or it is a branching node of $\text{cp}(T^\circ)$, or it is a branching node of (T'_v, r'_v). In all these cases, from the Step "updating K_v", $v_{att} \in K_v$ and $K_v = K_u \setminus V(P_u(v_{leaf}, v_{att})) \cup \{v_{att}\}$, where $P_u(v_{leaf}, v_{att})$ is the path between v_{leaf} and v_{att} in T_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}) \cup \{v_{att}\} = K_u \setminus V(P_u(v_{leaf}, v_{att})) \cup \{v_{att}\} = K_v$.

Hence, $K_s = K_v$.

Claim 7. *$r_s = r_v$, $dist_v = dist_s$.*

From the Char procedure, the root r_s is the vertex of T'_v that is closest to r° in T° and the root r_u is the vertex of T'_u that is closest to r° in T°. Since, T'_v is a subtree of T'_u, r_u is a vertex on the path between r° and r_s.

There are two cases to consider: (1) r_u is not an internal node on the path between v_{leaf} and v_{att}, or (2) r_u is an internal node on the path between v_{leaf} and v_{att}.

Case (1): r_u is not an internal node on the path between v_{leaf} and v_{att}. Then, since T'_v is obtained from T'_u by removing the internal vertices of $P(v_{leaf}, v_{att})$ and the vertex v_{leaf}, we have that $r_u = r_s$. Moreover, since $r_u = r_s$ we have $dist_s = dist_u$. Therefore, from Step "removing a from T_u", $r_v = r_u = r_s$ and $dist_v = dist_u = dist_s$.

Case (2): r_u is an internal node on the path between v_{leaf} and v_{att}, then $r_v = v_{att}$. The value $dist_u$ is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r° and r_u.

From Char procedure, $r_s = v_{att}$. The value $dist_s$ is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r° and r_s.

Let $p' = dist_s - dist_u$. p' is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r_u and $r_s = v_{att}$. That is, p' is the number of branching nodes of $\mathrm{cp}(T^\circ)$ nodes in the path between r_u and v_{att}. Since, from the Char procedure to obtain C_u, every branching node x of $\mathrm{cp}(T^\circ)$ receives label $branch_u(x) = 1$, we have that p' is the number of nodes x in the path between r_u and v_{att} such that $branch_u(x) = 1$. Therefore, $p' = p$, where p is the value obtained in step "preparation" of procedure *ForgetNode*. Therefore, from Step "removing a from T_u", $dist_v = p + dist_u = p' + dist_u = dist_s$.

Claim 8. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

Let $P_v(x, y)$ be a path of C_v such that x and y belongs to K_v and no other internal nodes of $P_v(x, y)$ belongs to K_v. Since $K_v = K_s$, let $P_s(x, y)$ be the corresponding path of C_s between x and y. Assume that the labels of $P_v(x, y)$ and the labels of $P_s(x, y)$ are given by the functions ℓ_v and ℓ_s respectively.

Since $K_v \subseteq K_u$, let $P_u(x, y)$ be the corresponding path of C_u between x and y and let $P^\circ(x, y)$ be the corresponding path in T° between x and y.

We want to show that $P_v(x, y) = P_s(x, y)$. That is, the paths $P_v(x, y)$ and $P_s(x, y)$ have the same sequence of vertices and their labels, given by functions ℓ_v and ℓ_s, are the same. There are two cases to consider, (1) v_{att} is not an internal node of $P_u(x, y)$ or (2) v_{att} is an internal node of $P_u(x, y)$.

Case (1): Assume that v_{att} is not an internal node of $P_u(x, y)$. From the Char procedure, $P_s(x, y) = \mathrm{Contr}(P^\circ(x, y))$ and $P_u(x, y) = \mathrm{Contr}(P^\circ(x, y))$. From the Step "contracting the paths" $P_v(x, y) = \mathrm{Contr}(P_u(x, y))$. Then, from item 1 of Lemma 16 (i.e., for any path P, $\mathrm{Contr}(P) = \mathrm{Contr}(\mathrm{Contr}(P))$) and $P_v(x, y) = \mathrm{Contr}(P_u(x, y)) = P_s(x, y)$.

Case (2): Assume that v_{att} is an internal node of $P_u(x, y)$. Consequently, $K_v = K_s = K_u \setminus \{v_{att}\}$. From the Char procedure we have that $P_s(x, y) = \text{Contr}(P^\circ(x, y))$ and $P_u(x, y) = \text{Contr}(P^\circ(x, v_{att}) \odot P^\circ(v_{att}, y))$. Then, from Step "contracting the paths" $P_v(x, y) = \text{Contr}(P_u(x, y))$. From item 3 of Lemma 16 (i.e., for any path $P_1 \odot P_2$ we have that $\text{Contr}(P_1 \odot P_2) = \text{Contr}(\text{Contr}(P_1) \odot \text{Contr}(P_2))$) $P_v(x, y) = P_s(x, y)$.

Therefore, for any pair of vertices x, y in $K_v = K_s$ such that the path between x and y has no vertices in $K_v = K_s$ we have that $P_v(x, y) = P_s(x, y)$. Hence, T_s is isomorphic to T_v and the labels of correspondent vertices of T_v and T_s are the same.

Claim 9. *$\sigma_v = \sigma_s$, $branch_v = branch_s$, $out_v = out_s$ and $father_v = father_s$.*

It is easy to check that $\sigma_v = \sigma_s$, since $T_s = T_v$ and σ_s can be obtained through σ_u by removing the mapping of v_{leaf} to a.

From Step "removing a from T_u" we have that $branch_v = branch_u$. Then, from the induction hypothesis, $branch_u(x) = 1$ if an only if x is a branching node of $\text{cp}(T^\circ)$ in T'_u. Since T'_v is a subtree of T'_u, $branch_u(x) = branch_s(x)$ for all $x \in V(T'_v)$.

Recall that $out_u(x)$ is the maximum number of branching nodes of $\text{cp}(T^\circ)$ in T'_u between x and a leaf of $T^\circ \setminus T'_u$ that does not have any internal node belonging to the set of vertices of T'_u and are not r'_u. Since T'_v is obtained from T'_u by removing the vertices of $V(P(v_{leaf}, v_{att})) \setminus \{v_{att}\}$, for any $x \neq v_{att}$ we have that $out_s(x) = out_u(x)$.

There are two cases to consider to show that $out_v(v_{att}) = out_s(v_{att})$: (1) r'_u is an internal node of $P(v_{att}, v_{leaf})$ in T'_u or (2) r'_u is not an internal node of $P(v_{att}, v_{leaf})$ in T'_u.

Case (1): If r'_u is an internal node of $P(v_{att}, v_{leaf})$, then $out_u(v_{att}) = out_s(v_{att})$. Then, step "removing a from T_u" ensures that $out_v(v_{att}) = out_u(v_{att})$.

Case (2): If r'_u is not an internal node of $P(v_{att}, v_{leaf})$, then $out_s(v_{att})$ is given by $\max\{out_u(v_{att}), \text{brheight}_T\, w\}$ where w is the neighbor of v_{att} in $P(v_{att}, v_{leaf})$. Then, step "removing a from T_u" ensures that $out_v(v_{att}) = \max\{out_u(v_{att}), \text{brheight}(w)\}$.

In both cases, we have $out_v(v_{att}) = out_s(v_{att})$.

For any vertex $x \in T'_u$, $father_s(x) = 1$ if and only if x has a non-leaf child in T°. Hence, $father_s(x) = father_u(x)$. Therefore, from the Step "removing a from T_u" we have $father_s(x) = father_v(x)$.

This proves that $C_v = C_s$. Hence, $C_v = \text{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$. Moreover, $C_v \in \text{FSC}_{\text{k,q}}(v)$ from step "updating $\text{FSC}_{\text{k,q}}(v)$".

Therefore this proves the lemma. For any q-branched partitioning tree $(T^\circ, r^\circ, \sigma^\circ)$ of A_v with Φ-width at most k we have that $C_v = \text{Char}((T^\circ, r^\circ, \sigma^\circ), X_v) \in \text{FSC}_{\text{k,q}}(V)$.

In other words, $\text{FSC}_{\text{k,q}}(v)$ is a full set of (k, q)-characteristics of A_v restricted to X_v. □

Theorem 29. *Procedure* ForgetNode *computes a full set of (k, q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity*

of procedure ForgetNode *is bounded by a a function* $f_f(k,q,k') = O\left((kqk')^3(60kqk')^{45kqk'}\right)$, *if* $q < \infty$, *or a function* $f'_f(k,k') = O\left((kk')^3(15kk')^{45kqk'}\right)$ *otherwise.*

Proof. From Theorem 28, procedure *ForgetNode* computes a full set of (k,q)-charac-teristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted.

For each element $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ steps 1 to 3 can be done in $O(|T_u|)$.

Contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges verifying if a contraction operation can be done between them. Hence, step 4 can be executed in $O(|T_u|^3)$.

Lastly, step 5 can be executed in $O(1)$.

Since the size and the number of elements in $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is bounded by $f'(k,q,k') = O(kqk')$ and $f(k,q,k') = O((60kqk')^{45kqk'})$ respectively from Lemma 21, we get the result. That is, the complexity of procedure *ForgetNode* is bounded by, if $q < \infty$:

$$f_f(k,q,k') = O\left(f(k,q,k') \cdot f'(k,q,k')^3\right) = O\left((kqk')^3(60kqk')^{45kqk'}\right).$$

If $q = \infty$ then the complexity of procedure *ForgetNode* is bounded by:

$$f_f(k,k') = O\left(f(k,k') \cdot f'(k,k')^3\right) = O\left((kk')^3(15kk')^{45kqk'}\right).$$

□

6.4 Procedure *JoinNode*

Let v be a join node of D, let u, w be its children, let $\mathrm{FSC}_{\mathrm{k,q}}(u)$ be a full set of characteristics of A_u restricted to X_u, and $\mathrm{FSC}_{\mathrm{k,q}}(w)$ a full set of characteristics of A_w restricted to X_w.

Remark 3. Procedure *JoinNode* tries to merge the (k,q)-characteristics for X_u and X_w that share a same structure, in contrast with the procedure Join Node from Section 4 that merges labeled partitioning trees for A_u and A_w that are isomorphic.

The *skeleton* $\mathrm{Sk}(C)$ of $C = ((T', r', \sigma'), \ell', K', dist', out', branch', father')$ is the tree obtained from T' by contracting all vertices that are not in K' (these vertices have degree two, thus the notion of contraction is well defined). Therefore, $V(\mathrm{Sk}(C)) = K'$. Two partitioning trees (T, r, σ) and (T', r', σ') are *isomorphic* if there is an one-to-one function $\varphi : V(T) \to V(T')$ preserving the edges, such that $\varphi(r) = r'$, and moreover, $\sigma'(\varphi(f)) = \sigma(f)$ for any leaf f of T.

The *structure* $\mathrm{Struct}(C)$ of a characteristic C is the partitioning tree obtained from $\mathrm{Sk}(C)$ by contracting all its vertices with degree two, different from the root. That is, we only keep branching nodes of T in $\mathrm{Struct}(C)$, while keeping the same root and the same mapping over the leaves of the tree. For any characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, dist_w, out_w, branch_w, father_w) \in \mathrm{FSC}_{\mathrm{k,q}}(w)$, with isomorphic structures and with $dist_w = 0$ (if both have distance non-zero we do not do the procedure *JoinNode* with C_u and C_w, since

the roots r_u and r_v come from different branches of the partitioning tree), Procedure *JoinNode* proceeds as follows, repeating the five steps below, for any possible execution of Step 2. Roughly, it merges C_u and C_w to obtain $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, dist_v, out_v, branch_v, father_v)$.

1. *identifying the structures:*

   To obtain T_v, we start by a copy of T_u and a copy of T_w.

   Then, for any vertex $t' \in V(\text{Struct}(C_u)) = V(\text{Struct}(C_w))$, let t_u be the corresponding vertex in T_u, and t_w be the corresponding vertex in T_w.

   In T_v, we identify t_u with t_w.

   Note that r_u and r_w are identified, let r_v be the resulting vertex.

   Then, since $dist_w = 0$, we set $dist_v \leftarrow dist_u$.

2. *merging the paths:*

   For any $\{x, y\} \in E(\text{Struct}(C_u))$, let x and y be vertices of T_u resulting of the identification of x_u with x_w and y_u with y_w respectively.

   Currently in T_v, there are two paths between x and y, a path P_u (initially a path of T_u) and a path P_w (initially a path of T_w), these paths are vertex-disjoint except for x and y. Since internal vertices of P_u and P_w do not belong to $V(\text{Struct}(C_u)$ nor to $V(\text{Struct}(C_w)$, any internal vertex (resp., edge) of both these paths defines the same partition $\mathcal{P}$ of X_v.

   Then, we replace P_u and P_w in T_v with a merging of P_u and P_w using the function $F : (i, j) \to H_\Phi(i, j, \mathcal{P})$.

3. *update of K_v:*

   Roughly, K_v is obtained by taking $K_u \cup K_w$ and some other vertices.

   Formally, starting from $K_v = \emptyset$.

   For any vertex x_v in T_v that results from the identification of $x_u \in V(T_u)$ and $x_w \in V(T_w)$ we set $K_v \leftarrow K_v \cup \{x_v\}$. In other words, x_u and x_w are either leaves of T_u and T_w or they are branching nodes of T_u and T_w, consequently they x_v is either a leaf or a branching node of T_v.

   For any other vertex x_v in $V(T_v)$, assume that x_v is obtained through the merging o a path P_u of T_u and a path P_w of P_w, as described in the step "merging of paths". Let x_u be the vertex of the extension of P_u used to generate x_v and x_w be the vertex of the extension of P_w used to generate x_v during the merging of P_u and P_w. Then, if $x_u \in K_u$ or $x_w \in K_w$, then we set $K_v \leftarrow K_v \cup \{x_v\}$.

4. *update of labels:*

   For any $x_v \in V(\text{cp}(T_v))$:

   $$\begin{aligned} branch_v(x_v) &\leftarrow \max\{branch_u(x_u), branch_w(x_w)\}, \\ out_v(x_v) &\leftarrow \max\{out_u(x_u), out_w(x_w)\}, \\ father_v(x_v) &\leftarrow \max\{father_u(x_u), father_w(x_w)\}. \end{aligned}$$

   For every $x_v \in V(\text{cp}(T_v))$, if $branch_u(x_u) = branch_w(x_w) = 0$ and $father_u(x_u) = father_w(x_w) = 1$, then $branch_v(x_v) \leftarrow 1$.

   For every $x_v \in V(T_v)$ such that x_v is a leaf, $\sigma_v(x_v) \leftarrow \sigma_u(x_u)$.

5. *contracting the paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

6. *update of* $\mathrm{FSC}_{\mathrm{k,q}}(v)$*:*

   $\mathrm{brheight}_T(r_v)$ is computable thanks to out_v and $branch_v$. If $dist_v + \mathrm{brheight}_T(r_v) \leq q$ and $\ell_v(t) \leq k$ for any internal vertex $t \in V(T_v)$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, then $\mathrm{FSC}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSC}_{\mathrm{k,q}}(v) \cup \{C_v\}$.

The rest of this section is dedicated to proving Lemma 30.

Lemma 30. JoinNode *computes a full set of* (k, q)*-characteristics of* A_v *restricted to* X_v*.*

Proof. Since $A_v = A_u \cup A_w$, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u and A_w do. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$ and $\mathrm{FSC}_{\mathrm{k,q}}(w) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$.

To prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of characteristics, we take any q-branched partitioning tree $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ of A_v with Φ width at most k and show that after procedure *JoinNode* finishes we have that $\mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v) \in \mathrm{FSC}_{\mathrm{k,q}}(v)$.

Roughly, we show that there is there is a particular execution of step 2 of procedure *JoinNode* with two characteristics, $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w \in \mathrm{FSC}_{\mathrm{k,q}}(w)$, resulting in C_v such that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v))$. A scheme of the proof of Lemma 30 can be found in Figure 8.

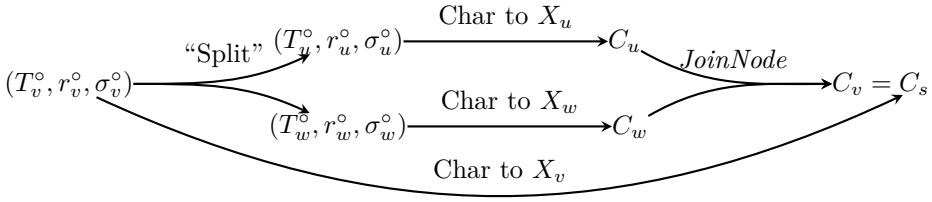


Figure 8: Scheme of proof of Lemma 30.

Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be any q-branched partitioning tree of A_v with Φ width at most k. Let $C_s = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. That is, $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, dist_s, out_s, branch_s, father_s)$ is the (k, q)-characteristic of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ restricted to X_v.

Let T'_v be the smallest subtree of T_v° spanning all leaves of T_v° that map elements of X_v. Let r'_v be the vertex of T'_v that is closest to r_v° in T_v°. That is, T'_v and r'_v are the tree and the vertex obtained with the first step of procedure $\mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

Let T_u° (resp. T_w°) be the smallest subtree of T_v° spanning all leaves of T_v° that map elements of A_u (resp. A_w) and let r_u° (resp. r_w°) be the vertex of T_u° (resp. T_w°) that is closest to r_v° in T_v°.

Since $(D, \mathcal{X})$ is a nice decomposition of A we have that $A_u \cap A_w = X_v$. Therefore, we have that $V(T_u^\circ) \cap V(T_w^\circ) = V(T'_v)$ and $E(T_u^\circ) \cap E(T_w^\circ) = E(T'_v)$.

Moreover, since $A_v = A_u \cup A_w$, we have that either ($r_u^\circ = r_v^\circ$ and $r_w^\circ = r'_v$) or ($r_w^\circ = r_v^\circ$ and $r_u^\circ = r'_v$). W.l.o.g. assume that $r_u^\circ = r_v^\circ$ and $r_w^\circ = r'_v$.

Then, $(T_u^\circ, r_u^\circ, \sigma_u)$, where σ_u is the restriction of σ_v over A_u, is a q-branched partitioning tree with Φ width at most k for A_u and $(T_w^\circ, r_w^\circ, \sigma_w)$, where σ_w is the restriction of σ_v over A_w, is a q-branched partitioning tree with Φ width at most k for A_w.

Let $C_u = \mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u), X_u)$ and $C_w = \mathrm{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$, such that $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathit{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u)$ and such that $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \mathit{dist}_w, \mathit{out}_w, \mathit{branch}_w, \mathit{father}_w)$. Since $(T_u^\circ, r_u^\circ, \sigma_u)$ and $(T_w^\circ, r_w^\circ, \sigma_w)$ are q-branched and with Φ width at most k, we have that $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w \in \mathrm{FSC}_{\mathrm{k,q}}(w)$. We want to show that there is an execution of procedure *JoinNode* on C_u and C_w generating C_v such that $C_v = C_s$.

In order to do that, we must first show that $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$ are isomorph and that either $\mathit{dist}_u = 0$ or $\mathit{dist}_w = 0$.

Claim 10. *Procedure* JoinNode *can be applied to C_u and C_w. That is,* $\mathrm{Struct}(C_u)$ *and* $\mathrm{Struct}(C_w)$ *are isomorph and either $\mathit{dist}_u = 0$ or $\mathit{dist}_w = 0$.*

From the fact that $X_v = X_u = X_w$, we have that, in the first step of $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u), X_u)$ and $\mathrm{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$ the smaller subtree of T_u° and T_w° spanning all vertices in $X_u = X_w = X_v$ is T'_v.

Since $r'_v = r_w^\circ$, from step 2 of the procedure $\mathrm{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$ we have that $\mathit{dist}_w = 0$.

We need to show that $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$ are isomorph. From the definition of structure, $\mathrm{Struct}(C_u)$ is obtained from $(\mathrm{Sk}(C_u), r_u, \sigma_u)$ by taking contracting all vertices of degree two that are different from the root from $\mathrm{Sk}(C_u)$. On the other hand, $\mathrm{Sk}(C_u)$ is the tree obtained from T_u by contracting all vertices that are not in K_u. Let $\mathrm{Struct}(C_u) = (T_u^r, r_u, \sigma_u)$ and $\mathrm{Struct}(C_u) = (T_w^r, r_w, \sigma_w)$, where T_u^r and T_w^r are the trees obtained by contracting all vertices of degree two different from the root (r_u and r_w respectively) from $\mathrm{Sk}(C_u)$ and $\mathrm{Sk}(C_w)$ respectively.

Since, r'_v is the vertex of T'_v that is closest to $r_u^\circ = r_v^\circ$ in T_u°, we have that $r_u = r'_v$. Then, since $r'_v = r_w$ we have that $r_u = r_w$.

Note that leaves of T'_v cannot be contracted, since they have degree one, in order to obtain T_u^r and T_w^r. In other words, if x is a leaf of T'_v we have that $x \in T_u^r$ and $x \in T_w^r$. Therefore, from the step one of Char procedure we have that, for any leaf $x \in V(T'_v)$, $\sigma_u(x) = \sigma_v^\circ(x) = \sigma_w(x)$.

Now we only need to show that the trees T_u^r and T_w^r are isomorph. In fact, we shall show that $T_u^r = T_w^r$.

Note that, from the steps one and six of the Char procedure, T_u (resp. T_w) is obtained from T'_v by applying some contraction operations on vertices that are not in K_u (resp. K_w), since they have degree two this is well defined. On the other hand, $\mathrm{Sk}(C_u)$ (resp. $\mathrm{Sk}(C_w)$) is the tree obtained from T_u by contracting all vertices with degree two that are not in K_u (resp. K_w). Hence, $\mathrm{Sk}(C_u)$ (resp. $\mathrm{Sk}(C_w)$) can be obtained from T'_v by contracting all vertices that are not in K_u (resp. K_w). Then, T_u^r (resp. T_v^r) is obtained from $\mathrm{Sk}(C_u)$ (resp. $\mathrm{Sk}(C_w)$) by contracting all vertices of degree two which are different from the root $r_u = r'_v$ (resp. $r_w = r'_v$).

Since, both T_u^r and T_w^r are obtained from T'_v by contracting some vertices with degree two, we only need to show that the same vertices of T'_v are contracted to obtain T_u^r and T_v^r. That is, we only need to show that $V(T_u^r) = V(T_v^r)$.

In order to do that consider the following cases for $x \in V(T'_v) \setminus \{r'_v\}$: (1) $x \notin K_u \cup K_w$, (2) $x \in K_u \setminus K_w$, (3) $x \in K_w \setminus K_u$ and (4) $x \in K_u \cap K_w$.

Case 1: If $x \notin K_u \cup K_w$ then x has degree two in T'_v and it is contracted to obtain T^r_u and T^r_w. Hence, $x \notin V(T^r_u)$ and $x \notin V(T^r_w)$.

Case 2: If $x \in K_u \setminus K_w$, then x is not a leaf of T'_v, is not the parent of a leaf in T'_v, is not a branching node of T'_v and it is not a branching node of $\mathrm{cp}(T^\circ_w)$, otherwise x would be in K_w. Therefore, x has degree two in T'_v. Since, x is not r'_v we have that x is contracted in the process to obtain T^r_u from $\mathrm{Sk}(C_u)$. In other words, $x \notin V(T^r_u)$. On the other hand, $x \notin K_w$, hence $x \notin V(\mathrm{Sk}(C_w))$ and, consequently, $x \notin V(T^r_w)$.

Case 3: This case is similar to Case 2 with the role of K_u and K_w reversed. If $x \in K_w \setminus K_u$, then x is not a leaf of T'_v, is not the parent of a leaf in T'_v, is not a branching node of T'_v and it is not a branching node of $\mathrm{cp}(T^\circ_u)$, otherwise x would be in K_u. Therefore, x has degree two in T'_v. Since, x is not r'_v we have that x is contracted in the process to obtain T^r_w from $\mathrm{Sk}(C_w)$. In other words, $x \notin V(T^r_w)$. On the other hand, $x \notin V(\mathrm{Sk}(C_u))$ and, consequently $x \notin V(T^r_u)$.

Case 4: If $x \in K_u \cap K_w$, then $x \in \mathrm{Sk}(C_u)$ and $x \in \mathrm{Sk}(C_w)$. Moreover, either x is a leaf of T'_v, is the parent of a leaf in T'_v, is a branching node of T'_v, or is a branching node of $\mathrm{cp}(T^\circ_u)$ and $\mathrm{cp}(T^\circ_w)$.

If x is a leaf of T'_v, then x has degree one in T'_v and, hence, $x \in V(T^r_u)$ and $x \in V(T^r_w)$.

If x is not a leaf of T'_v, then x has degree two in $\mathrm{Sk}(C_u)$ if an only if x has degree two in $\mathrm{Sk}(C_w)$. This is due to the fact that, by contracting vertices of degree two of a tree, we do not change the degrees of the remainder vertices.

This shows that $V(T^r_u) = V(T^r_w)$. From the fact that T^r_u and T^r_w are both obtained from T'_v by contraction of vertices of degree two we get the result. That is, $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$ are isomorph.

Therefore, the there is an execution of procedure *JoinNode* were C_u is merged with C_w. Let $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathit{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v)$ be the result of a particular execution of procedure *JoinNode* on C_u and C_w, that will be explained latter in this proof.

We want to show that $C_v = C_s$.

How C_v is obtained.

We need to specify how procedure *JoinNode* merges the paths in T_u with the paths in T_w. That is, we need to specify how step "merging the paths" proceeds to merge the paths of T_u and T_w to obtain T_v. In order to do that, we first show how paths in T°_v with labels given by the function Φ can be seen as mergings of the corresponding paths in T°_u and T°_w.

Let $\{x, y\}$ be any edge in $E(\mathrm{Struct}(C_u)) = E(\mathrm{Struct}(C_w))$. We have that, in T'_v, all the internal vertices and edges on the path $P'_v(x, y)$ between x and y define the same partition $\mathcal{T}$ of X_v, since the vertices have degree two. Let $P^\circ_v(x, y)$, $P^\circ_u(x, y)$ and $P^\circ_w(x, y)$ be the paths between x and y in T°_v, T°_u and T°_w respectively. Note that the internal vertices and edges of these paths define the same partition $\mathcal{T}$ of X_v.

For any internal vertex or edge x' of $P_v^\circ(x,y)$ let $\mathcal{P}(x')$ be the partition of A_v defined by x'. Then, in T_u° and in T_w°, the vertex or edge x' defines the partition $\mathcal{P}(x') \cap A_u$ of A_u and $\mathcal{P}(x') \cap A_w$ of A_w respectively. Since Φ is compatible with $(D, \mathcal{X})$, we have that there is a function H_Φ such that for any partition $\mathcal{P}$ of A_v it is true that $\Phi_{A_v}(\mathcal{P}) = H_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \Phi_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v)$. Therefore, for any internal vertex or edge x' of $P_v^\circ(x,y)$ we have that $\Phi_{A_v}(\mathcal{P}(x')) = H_\Phi(\Phi_{A_u}(\mathcal{P}(x') \cap A_u), \Phi_{A_w}(\mathcal{P}(x') \cap A_w), \mathcal{T})$. Hence, the labeled path $P_v^\circ(x,y)$ with labels given by the function Φ can be obtained by the merging of the path $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$, both with labels given by Φ, under the function $F(i,j) = H_\Phi(i,j,\mathcal{T})$. Since $|V(P_u^\circ(x,y))| = |V(P_w^\circ(x,y))|$, the extensions of $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ used in the merging are simply $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ themselves. In other words, we can merge the paths $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ under the function F to obtain the path $P_v^\circ(x,y)$.

Let $P_u(x,y)$ (resp. $P_w(x,y)$) be the path between x and y in T_u (resp. T_v). In step "merging the paths" of procedure *JoinNode*, the path $P_u(x,y)$ is merged with the path $P_w(x,y)$. From the Char procedure, we have that $P_u(x,y)$ (resp. $P_w(x,y)$) is obtained from $P_u^\circ(x,y)$ (resp. $P_w^\circ(x,y)$) by applying some contraction operations. Then, let $P_v(x,y)$ be a merging of $P_u(x,y)$ with $P_w(x,y)$ under the function F such that $P_v^\circ(x,y)$ *respects*[2] $P_v(x,y)$. Roughly, this means that the vertices and edges of $P_v(x,y)$, which is a merging of $P_u(x,y)$ and $P_w(x,y)$ under the function F, have "equivalent" vertices in $P_v^\circ(x,y)$, which is a merging of $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ under the same function F.

Since procedure *JoinNode* tries all possible ways of merging $P_u(x,y)$ and $P_w(x,y)$ for all $\{x,y\} \in E(\text{Struct}(C_u))$, we have that there is an execution of procedure *JoinNode* where for all $\{x,y\} \in E(\text{Struct}(C_u))$ the path $P_v(x,y)$ obtained through the merging of $P_u(x,y)$ and $P_w(x,y)$ under F is respected by $P_v^\circ(x,y)$.

Let T_v be the tree obtained by such execution of procedure *JoinNode*.

To show that C_v, obtained through this particular execution of procedure *JoinNode*, is equal to $C_s = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we start by showing that T_v is isomorph to T_s.

Claim 11. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

In the following consider that the labels of a tree are given by either the associated function ℓ or by Φ if the tree has no associated function ℓ. That is, the labels of T_u, T_w, T_s and T_v are given by ℓ_u, ℓ_w, ℓ_s and ℓ_v, while the labels of T_v°, T_v', T_u° and T_w° are given by the function Φ.

Note that from the fact that T_v is obtained by merging paths $P_u(x,y)$ and $P_w(x,y)$ for every edge $\{x,y\} \in E(\text{Struct}(C_u))$ we have that the tree obtained by contracting all vertices of degree two in T_v is isomorph to the tree in $\text{Struct}(C_u)$.

As explained above, for any edge $\{x,y\} \in E(\text{Struct}(C_u))$ to obtain T_v we first merge the paths $P_u(x,y)$ and $P_w(x,y)$ obtaining $P_v(x,y)$ and then we apply some contraction operations on $P_v(x,y)$ (step "contracting the paths") obtaining $P_v^c(x,y)$. More precisely, let $(x_1, \ldots, x_z)$ be the sequence of vertices in the path from x to y in the tree obtained immediately after step "merging the paths" of procedure *JoinNode* that are in K_v. That is, $P_v(x,y)$ can be

[2]For the definition of "respect" see "Merging of Labeled Paths" in Section 5.

written as $P_v(x,x_1) \odot P_v(x,x_2) \odot \cdots \odot P_v(x_z,y)$ where $x_i \in K_v$, for $1 \leq i \leq z$. Then, $P_v(x,y)$ is replaced by $\mathrm{Contr}(P_v(x,x_1)) \odot \mathrm{Contr}(P_v(x_1,x_2)) \odot \cdots \odot \mathrm{Contr}(P_v(x_z,y))$. Let $P_v^c(x,y) = \mathrm{Contr}(P_v(x,x_1)) \odot \mathrm{Contr}(P_v(x,x_2)) \odot \cdots \odot \mathrm{Contr}(P_v(x_z,y))$. Hence, for each $\{x,y\} \in E(\mathrm{Struct}(C_u))$ we have that $P_v^c(x,y)$ is the path in T_v between x and y.

Note that, $K_s = K_u \cup K_w$. Since, the only case where $x \in K_s$ and $x \notin K_u$ is when x is not a leaf of T'_v, nor the parent of a leaf of T'_v, nor a branching node of $\mathrm{cp}(T_u^\circ)$, but it is a branching node of $\mathrm{cp}(T_v^\circ)$. Therefore, x is is a branching node of $\mathrm{cp}(T_w^\circ)$ and, hence, $x \in K_w$.

On the other hand, the path $P_s(x,y)$ between x and y in C_s is obtained from $P_v^\circ(x,y)$ by applying some contraction operations. That is, let $(x_1^s,\ldots,x_z^s)$ be the sequence of vertices in the path from x to y in T_v° that are in K_s, then $P_s(x,y)$ can be written as $\mathrm{Contr}(P_v^\circ(x,x_1^s)) \odot \mathrm{Contr}(P_v^\circ(x_1^s,x_2^s)) \odot \cdots \odot \mathrm{Contr}(P_v^\circ(x_z^s,y))$.

Then, this proof essentially follows from Lemma 18 when applied to $P_v^\circ(x,y)$ and $P_v(x,y)$. That is, we consider $P_v^\circ(x,y)$ as M where P and Q are $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$, respectively. $P_v(x,y)$ takes the role as M^c where P^c and Q^c are $P_u(x,y)$ and $P_w(x,y)$ respectively. Finally, we set K_p as the set K_u and K_q as the set K_w. Then, Lemma 18 guarantees that $\mathrm{Contr}(P_v(x,x_1)) = \mathrm{Contr}(P_v^\circ(x,x_1^s))$, $\mathrm{Contr}(P_v(x_z,y)) = \mathrm{Contr}(P_v^\circ(x_z^s,y))$ and, for all $1 \leq i \leq z-1$, $\mathrm{Contr}(P_v(x_i,x_{i+1})) = \mathrm{Contr}(P_v^\circ(x_i^s,x_{i+1}^s))$. Therefore, since $P_s(x,y) = \mathrm{Contr}(P_v^\circ(x,x_1^s)) \odot \mathrm{Contr}(P_v^\circ(x_1^s,x_2^s)) \odot \cdots \odot \mathrm{Contr}(P_v^\circ(x_z^s,y))$ and $P_v^c(x,y) = \mathrm{Contr}(P_v(x,x_1)) \odot \mathrm{Contr}(P_v(x,x_2)) \odot \cdots \odot \mathrm{Contr}(P_v(x_z,y))$, we get the result. That is, $P_v^c(x,y) = P_s(x,y)$.

Therefore T_v and T_s are isomorph and for all internal vertices or edges t_v of T_v with a correspondent vertex or edge t_s of T_s we have that $\ell_v(t_v) = \ell_s(t_s)$.

Since T_v and T_s are isomorph, for every vertex $x_s \in T_s$, let x_v be its correspondent in T_v. To make the rest of this proof easier to read, we abuse the notation to say that $x_s = x_v$. We now prove that $r_v = r_s$ and that $K_v = K_s$.

Claim 12. *$r_v = r_s$ and $K_v = K_s$.*

From step "identifying the structures" we have that $r_v = r_u$. We have that $r_u = r'_v$ from step one of the Char procedure to obtain C_u. Then, by the fact that $r_s = r'_v$ from step one of the Char procedure to obtain C_s, we have that $r_s = r_v$.

Let x_s be a vertex of T_s that is not the root of T_s. Since T_v is isomorph to T_s, set x_v be the corresponding vertex of x_s in T_v. To show that $K_s = K_v$, there are a few cases to consider: (1) x_s is a leaf of T'_v, (2) x_s is a branching node of T'_v, (3) x_s is the parent of a leaf in T'_v, (4) x_s is a branching node of $\mathrm{cp}(T_v^\circ)$, (5) otherwise. That is, cases (1) to (4) are all the cases when $x_s \in K_s$, while, in case (5), $x_s \notin K_s$. We want to show that $x_s \in K_s$ if and only if $x_v \in K_v$.

Case (1): If x_s is a leaf of T'_v, then x_s has degree one in T'_v. Therefore, x_s is a vertex of $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$. Then, during step "identifying the structures" x_v is obtained through the identification of $x_s \in V(\mathrm{Struct}(C_u))$ and $x_s \in V(\mathrm{Struct}(C_w))$. Hence, during step "update of K_v" we have that x_v is put into K_v. That is, $x_v \in K_v$.

Case (2): If x_s is a branching node of T'_v, then x_s has degree at least three in T'_v. Therefore, x_s is a vertex of $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$. Then, during step "identifying the structures" x_v is obtained through the identification

of $x_s \in V(\text{Struct}(C_u))$ and $x_s \in V(\text{Struct}(C_w))$. Hence, during step "update of K_v" we have that x_v is put into K_v. That is, $x_v \in K_v$.

Case (3): If x_s is the parent of a leaf in T'_v and its not a branching node of T'_v, then x_s has degree two in T'_v. Since x_s has degree two in T'_v we have that $x_s \notin V(\text{Struct}(C_u))$. Therefore, x_v obtained during step "merging the path". That is, x_v is obtained through the merging of two vertices $x_u \in T_u$ and $x_w \in T_w$.

Let $\{x, y\} \in E(\text{Struct}(C_u))$ be two vertices such that x_s is an internal vertex on the path $P^\circ_v(x, y)$ from x to y in T°_v. Then, x_v is obtained through the merging $P_u(x, y)$ and $P_w(x, y)$. Let $P_v(x, y)$ be the resulting path. Recall that $P^\circ_v(x, y)$ can be written as a merging of the paths $P^\circ_u(x, y)$ and $P^\circ_w(x, y)$. From the construction of C_v we have that $P_v(x, y)$ is respected by $P^\circ_v(x, y)$.

Then, $x_s \in V(T^\circ_v)$ is the result of matching the vertex $x_u = x_s \in T^\circ_u$ with the vertex $x_w = x_s \in T^\circ_w$. On the other hand, since $P_v(x, y)$ is respected by $P^\circ_v(x, y)$, we have that x_v is obtained through the merging of $x_u \in T_u$ and $x_w \in T_w$.

From the Char procedure, since x_s is the parent of a leaf in T'_v, we have that $x_u \in K_u$ and $x_w \in K_w$. Then, since $x_u \in K_u$ and $x_w \in K_w$, step "update of K_v" ensures that $x_v \in K_v$.

Case (4): If x_s is a branching node of $\text{cp}(T^\circ_v)$ and x_s is not the parent of a leaf in T'_v and its not a branching node of T'_v, then x_s has degree two in T'_v and x_s has a non leaf child in T'_v. Since x_s is a branching node of $\text{cp}(T^\circ_v)$ and has degree two in T'_v, x_s has at least one non leaf child in $V(T^\circ_v) \setminus V(T'_v)$. Let x'_s be this child. Therefore, from the fact that $V(T^\circ_v) = V(T^\circ_u) \cup V(T^\circ_w)$ we have that x'_s is either in $V(T^\circ_u)$ or in $V(T^\circ_w)$. W.l.o.g. assume that $x'_s \in V(T^\circ_u)$, hence x'_s is a branching node of $\text{cp}(T^\circ_u)$. Consequently, from the Char procedure to obtain C_v, $x'_s \in K_u$. Note that, from step "update of K_v" if x_v is obtained by merging (or matching) a vertex x_u and x_w, then $x_v \in K_v$ if either $x_u \in K_u$ or $x_w \in K_w$. Then, the rest of this proof is similar to Case (3) and thus omitted.

Case (5): If $x_s \notin K_s$, then x_s has degree two in T'_v, $x_s \notin K_u$ and $x_s \notin K_w$. Hence, following the same reasoning in Case (3), we have that x_v is obtained through the merging (or matching) of $x_u \notin K_u$ and $x_w \notin K_w$. Therefore, from step "update of K_v" we have that $x_v \notin K_v$.

Therefore, $x_v \in K_v$ if and only if $x_s \in K_s$.

Considering the previous claims, we only need to prove the following claim in order to show that $C_v = C_s$.

Claim 13. *$dist_s = dist_v$.*
For all leaves $x_s \in V(T_s)$ we have that $\sigma(x_v) = \sigma(x_s)$.
For all vertices $x_s \in V(\text{cp}(T_s))$ we have that:

$$out_s(x_s) = out_v(x_v);$$
$$branch_s(x_s) = branch_v(x_v);$$
$$father_s(x_s) = father_v(x_v).$$

From step "identifying the structures" we have that $dist_v = dist_u$. Note that $dist_s$ is the number of branching nodes of T_v° between r_v° and r'_v. On the other hand, $dist_u$ is the number of branching nodes of T_u° between r_u° and r'_v. Since $r_u^\circ = r_v^\circ$ and the path $P_v^\circ(r_v^\circ, r'_v)$, the path between r_v° and r'_v in T_v°, is equal to the path $P_u^\circ(r_u^\circ, r'_v)$, the path between r_u° and r'_v in T_u°, we get that $dist_u = dist_s$. Therefore, $dist_v = dist_s$.

Let x_s be a leaf of T_s and x_v be its corresponding vertex in T_v. Since x_s is a leaf of T_s, it is also a leaf in T'_v. Note that, from the definition of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ and $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$, we have that $\sigma_v^\circ(x_s) = \sigma_u^\circ(x_s)$. On the one hand, we have that, from the fact that $C_s = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, $\sigma_s(x_s) = \sigma_v^\circ(x_s)$. On the other hand, from the $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u = X_v)$, we have that $\sigma_u(x_s) = \sigma_u^\circ(x_s)$. Therefore, $\sigma_u(x_s) = \sigma_s(x_s)$. Note that, since x_s is a leaf in T'_v, $x_s \in V(\text{Struct}(C_u) = V(\text{Struct}(C_w)))$. Therefore, x_v is obtained through the identification of $x_s = x_u \in V(\text{Struct}(C_u)$ and $x_s = x_w \in V(\text{Struct}(C_w))$. Then, during step "update of labels" of procedure *JoinNode*, $\sigma_v(x_v)$ is set to be $\sigma_u(x_u = x_s)$. Hence, we get the result. That is, for every leaf $x_s \in V(T_s)$ we have that $\sigma(x_v) = \sigma(x_s)$.

Let x_s be a vertex of $\text{cp}(T_s)$. That is, x_s is an internal vertex of T_s. Let x_v be its correspondent vertex in T_v with x_u and x_w being the vertices of T_u and T_w used to create x_v. In other words, x_v is obtained from the merging of x_u and x_w or from the identification of x_u with x_w.

In C_s, $out_s(x_s)$ is the maximum number of branching nodes in a path of T_v° between x_s and a leaf in $V(T_v^\circ) \setminus V(T'_v)$ with no internal vertices belonging to $V(T'_v)$. Then, let $P(x_s, l)$ be any path of T_v° between x_s and a leaf in $V(T_v^\circ) \setminus V(T'_v)$ such that the number of branching nodes in this path is maximum and with no internal vertex of $P(x_s, l)$ belonging to T'_v. We have that $V(P(x_s, l)) \setminus \{x_s\}$ is entirely contained in $V(T_v^\circ) \setminus V(T'_v)$.

Since T_u° and T_w° are subtrees of T_v° such that $V(T_u^\circ) \cup V(T_w^\circ) = V(T_v^\circ)$, we have that either $l \in V(T_u^\circ)$ or $l \in V(T_v^\circ)$, but not both since $l \notin V(T'_v) = V(T_u^\circ) \cap V(T_w^\circ)$. Therefore, $V(P(x_s, l)) \setminus \{x_s\} \subseteq V(T_u^\circ) \setminus V(T'_v)$ or $V(P(x_s, l)) \setminus \{x_s\} \subseteq V(T_w^\circ) \setminus V(T'_v)$. That is, if $l \in V(T_u^\circ)$, then $P(x_s, l)$ is a path of T_u° that starts in x_s does not pass through any vertex in T'_v and ends in l, otherwise $P(x_s, l)$ is a path of T_w° that starts in x_s does not pass through any vertex in T'_v and ends in $l \in V(T_w^\circ)$.

If $l \in V(T_u^\circ)$, from the fact that $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u = X_v)$, then $out_u(x_u) = out_s(x_s)$. If $l \in V(T_w^\circ)$, from the fact that $C_w = \text{Char}((T_w^\circ, r_w^\circ, \sigma_w^\circ), X_w = X_v)$, then $out_w(x_w) = out_s(x_s)$.

Moreover, for all leaves $l \in V(T_u^\circ) \setminus V(T'_v)$, the paths $P(x_s, l)$ in T_u° such that $P(x_s, l)$ has no internal vertex in T'_v are all paths in T_v° that do not pass through any vertex in T'_v. Hence, $out_u(x_u) \leq out_s(x_s)$.

Similarly, for all leaves $l \in V(T_u^\circ) \setminus V(T'_v)$, the paths $P(x_s, l)$ in T_u° such that $P(x_s, l)$ has no internal vertex in T'_v are all paths in T_v° that do not pass through any vertex in T'_v. Hence, $out_w(x_w) \leq out_s(x_s)$.

Therefore, from step "update of labels", we have that $out_v(x_v) = \max\{out_u(x_u), out_w(x_w)\} = out_s(x_s)$.

If $branch_s(x_s) = 1$, then x_s is a branching node of $\text{cp}(T_v^\circ)$. Therefore, either (1) x_s is a branching node in $\text{cp}(T_u^\circ)$, (2) x_s is a branching node in $\text{cp}(T_w^\circ)$, or (3) x_s has exactly one non leaf child in T_u° and exactly one non leaf child in T_w°.

Case 1: If x_s is a branching node in $\mathrm{cp}(T_u^\circ)$, then $x_s \in K_s \cap K_u$. Hence, x_s is not contracted during $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ in order to obtain T_u, that is $x_u = x_s \in V(T_u)$.

To obtain T_v we merge the paths in T_u and T_w in such a way that each path merged is respected by the corresponding path in T'_v. Hence, x_v is obtained through the merging of $x_u = x_s$ and x_w. Since, from $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$, $branch_u(x_u) = 1$ and $branch_v(x_v) = \max\{branch_u(x_u), branch_w(x_w)\}$. Since $branch_w(x_w) \leq 1$, we have that $branch_v(x_v) = branch_s(x_s)$.

Case 2: Similar to case (1), by exchanging the roles of x_u and x_w, and thus omitted.

Case 3: If x_s has exactly one non leaf child in T_u° and exactly one non leaf child in T_w°. Therefore, by the Char procedure, we have that $father_u(x_s = x_u) = 1$ and $father_w(x_s = x_w) = 1$. Then, we have that $father_u(x_u) = 1$ and $father_w(x_w) = 1$. Hence, during step "update of labels", $branch_v(x_v)$ receives the value 1, since $branch_u(x_u) = branch_w(x_w) = 0$ and $father_u(x_u) = father_w(x_w) = 1$.

If $branch_s(x_s) = 0$, then x_s is not a branching node of $\mathrm{cp}(T_v^\circ)$. Hence, x_s is neither a branching node of $\mathrm{cp}(T_u^\circ)$ nor a branching node of $\mathrm{cp}(T_w^\circ)$. Moreover, x_s has no non leaf children in T_u° and no non leaf children in T_w°. Therefore, $branch_u(x_s = x_u) = branch_w(x_s = x_w) = 0$ and $father_u(x_s = x_u) = father_w(x_s = x_w) = 0$. Then, during step "update of labels", $branch_v(x_v) = \max\{branch_u(x_u), branch_w(x_w)\} = 0$.

Hence, we get the result. That is, for all $x_s \in V(\mathrm{cp}(T_s))$, we have that $branch_v(x_v) = branch_s(x_s)$.

Now it remains to show that for all $x_s \in V(\mathrm{cp}(T_s))$ we have that $father_v(x_v) = father_s(x_s)$.

This proof is similar to the proof that if $branch_s(x_s) = branch_v(x_v) = 0$, since $father_s(x_s) = 0$ implies that x_s has no non leaf children in $V(T_v^\circ) \setminus V(T'_v)$. Hence, x_s has no non leaf children in $V(T_u^\circ) \setminus V(T'_v)$ nor in $V(T_w^\circ) \setminus V(T'_v)$.

If $father_s(x_s) = 0$, then x_s is not a branching node of $\mathrm{cp}(T_v^\circ)$. Hence, x_s is neither a branching node of $\mathrm{cp}(T_u^\circ)$ nor a branching node of $\mathrm{cp}(T_w^\circ)$. Moreover, x_s has no non leaf children in T_u° and no non leaf children in T_w°. Therefore, $father_u(x_u) = father_w(x_w) = 0$. Then, during step "update of labels", $father_v(x_v) = \max\{father_u(x_u), father_w(x_w)\} = 0$.

If $father_s(x_s) = 1$, then x_s has either a non leaf child in $V(T_u^\circ) \setminus V(T'_v)$ or a non leaf child in $V(T_w^\circ) \setminus V(T'_v)$. Hence, $father_u(x_u) = 1$ or $father_w(x_w) = 1$. Then, during step "update of labels", $father_v(x_v) = \max\{father_u(x_u), father_w(x_w)\} = 1$.

Hence, we get the result. That is, for all $x_s \in V(\mathrm{cp}(T_s))$, we have that $father_v(x_v) = father_s(x_s)$.

This concludes the proof that C_v obtained through this execution of procedure *Join-Node* on $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w \in \mathrm{FSC}_{\mathrm{k,q}}(w)$ is such that $C_v = C_s = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

It remains to show that C_v is put in $\mathrm{FSC}_{\mathrm{k,q}}(v)$ by procedure *JoinNode*. By Lemma 20, C_v is a (k, q)-characteristic of A_v restricted to X_v. Hence, $\mathrm{brheight}(r_v) + dist_v \leq q$ and for all internal vertex or edge x_v of T_v we have

that $\ell_v(x) \leq k$. Consequently, during step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$" we have that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$. Therefore, $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of (k,q)-characteristics of A_v restricted to X_v. □

Theorem 31. *Procedure* JoinNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity of procedure* JoinNode *is bounded by a function*

$$f_j(k,q,k') = O\left((60kqk')^{45kqk'} \cdot 2^{O\left(\sqrt{kqk' \cdot \log(kqk')}\right)} \cdot (kqk')^{kqk'}\right), \text{ if } q < \infty,$$

or by a function

$$f'_j(k,k') = O\left((15kk')^{45kk'} \cdot 2^{O\left(\sqrt{kk' \cdot \log(kk')}\right)} \cdot (kk')^{kk'}\right), \text{ if } q = \infty.$$

Proof. From Theorem 30, procedure *JoinNode* computes a full set of (k,q)-characteristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted. From the fact that Φ is compatible with $(D, \mathcal{X})$, H_Φ can be computed in constant time.

For each pair of elements $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathit{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \mathit{dist}_w, \mathit{out}_w, \mathit{branch}_w, \mathit{father}_w) \in \mathrm{FSC}_{\mathrm{k,q}}(w)$, discovering if their structures are isomorph can be done in $2^{O(\sqrt{|T_u| \log |T_u|})}$ [BL83].

Then, for each pair of characteristics such that their structures are isomorph, r_u is the vertex correspondent to r_w and with $\mathit{dist}_u = 0$ we apply steps 1 to 6 of the *JoinNode* procedure.

Step 1 has complexity bounded by $O(|T_u|)$. Since the merging of two paths P and Q has size limited to $O(|P||Q|)$ by definition of merging, one execution of Step 2 takes at most $O(|T_u||T_w|)$ time. Steps 3 and 4, for each execution of Step 2, takes $O(|T_u||T_w|)$ time.

Since contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges verifying if a contraction operation can be done between them, Step 5 can be executed in $O((|T_u||T_w|)^3)$, for each execution of Step 2. Lastly, Step 6 can be executed in $O(|T_u||T_w|)$, for each execution of Step 2.

For any pair of paths P and Q merged during step 2 (merging the paths), let $p = \max |P|, |Q|$. Hence, there are at most $O(p^p)$ different ways of merging P and Q. That is, for each edge of P there are at most $|Q|$ edges in Q that is a possible match for P. Therefore, we can upper bound the amount of different possible executions of Step 2 by $O(\max\{|T_u|^{|T_u|}, |T_w|^{|T_w|}\})$. Note that, since $\max\{|T_u|, |T_w|\} \leq f'(k,q,k')$, we have that $O(\max\{|T_u|^{|T_u|}, |T_w|^{|T_w|}\}) \leq O(f'(k,q,k')^{f'(k,q,k')})$.

Since the size and the number of elements in $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is bounded by $f'(k,q,k') = O(kqk')$ and $f(k,q,k') = O((60kqk')^{45kqk'})$ respectively from Lemma 21, we get the result.

That is, complexity of procedure *JoinNode* is bounded by

$$f_j(k,q,k') = O\left(f\left(k,q,k'\right) \cdot 2^{O\left(\sqrt{f'(k,q,k') \cdot \log f'(k,q,k')}\right)} \cdot f'\left(k,q,k'\right)^{f'\left(k,q,k'\right)}\right),$$

$$f_j(k,q,k') = O\left((60kqk')^{45kqk'} \cdot 2^{O\left(\sqrt{kqk' \cdot \log(kqk')}\right)} \cdot (kqk')^{kqk'}\right).$$

With a similar proof for the case that $q = \infty$, we have:

$$f'_j(k,k') = O\left(f\left(k,k'\right) \cdot 2^{O\left(\sqrt{f'(k,k') \cdot \log f'(k,k')}\right)} \cdot f'\left(k,k'\right)^{f'\left(k,k'\right)}\right).$$

From Lemma 21, $f'(k,k') = O(kk')$ and $f(k,k') = O((15kk')^{45kk'})$, then

$$f'_j(k,k') = O\left((15kk')^{45kk'} \cdot 2^{O\left(\sqrt{kk' \cdot \log(kk')}\right)} \cdot (kk')^{kk'}\right).$$

□

6.5 Remarks and Structural Properties

Having shown the algorithm we can, now, provide the proof of Claim 1 which is: "For any partition function f compatible with a nice decomposition of some set A, the partition function *maxf* is also compatible".

Proof. Let $(D, \mathcal{X})$ be a nice decomposition of A with width not bigger than k'. Since f is compatible with $(D, \mathcal{X})$ we have that there are functions F_f and H_f such that for any partition $\mathcal{P}$ of A:

$$f_{A_v}(\mathcal{P}) = F_f(f_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u), \text{ and}$$
$$f_{A_v}(\mathcal{P}) = H_f(f_{A_u}(\mathcal{P} \cap A_u), f_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

To show that *maxf* is compatible with $(D, \mathcal{X})$ we have to show that there are function F_{maxf} and H_{maxf} that play the same role as F_f and H_f.

Clearly, if $\mathcal{A} = \{A_1, A_2\}$ is a bipartition of A then $maxf(\mathcal{A}) = f(\mathcal{A})$, since $\{f(\{A_1, A_2\}) = f(\{A_2, A_1\})$. Hence, for any bipartition $\mathcal{P}$ of A, the function F_{maxf} takes the same value as F_f and the function H_{maxf} takes the same value as H_f.

To show how to compute F_{maxf}, during the processing of an introduce node $v \in D$ with child u with $A_v \setminus A_u = \{a\}$, consider the step "update of labels of vertex(s) and edge(s)" of procedure *IntroduceNode* when applied to a characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ by subdividing an edge $f = \{v_{top}, v_{bottom}\}$.

Let $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ be a partitioning tree of A_u such that $C_u = \mathrm{Char}((T_u, r_u, \sigma_u), X_u)$. Let f_u° be the representative of f_u in T_u°. Then, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is the partitioning tree for A_v obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by subdividing the edge $f_u^\circ = \{v_{top}^\circ, v_{bottom}^\circ\}$ one time, creating a vertex v_{att}, and adding v_{leaf} as neighbor of v_{att}, make $\sigma_v^\circ(v_{leaf}) = a$ and $r_v^\circ = r_u^\circ$.

For each edge e of T_v let $\mathcal{T}_e$ ($\mathcal{T}_e^\circ$) be the partition of X_v (A_v) that it defines. Similarly, for each vertex t of T_v let $\mathcal{T}_t$ ($\mathcal{T}_t^\circ$) be the partition of X_v (A_v) that it defines.

Since an edge of T_v defines a bipartition of A_v, we can update the labels of edges in T_v using F_f, i.e. we apply the instruction "$\ell_v(e) \leftarrow F_f(\ell_u(e), \mathcal{T}_e, a)$" to edges of T_v. After this instruction, $\ell_v(e) = F_f(\ell_u(e), \mathcal{T}_e, a) = F_{maxf}(\ell_u(e), \mathcal{T}_e, a) = maxf(\mathcal{T}_e^\circ)$ for all edges of T_v.

For each internal vertex t of T_v, let E_t be the set of edges incident to t in T_v and let E_t° be the set of edges incident to t in T_v°. Assume, by induction, that $\ell_u(t) = maxf(\mathcal{T}_t^\circ \cap A_u)$.

From the fact that T_v° is a partitioning tree of A_v, we have that $\mathcal{T}_t^\circ = \{A_1, A_2, \ldots, A_{|E_t^\circ|}\}$ and, from the definition of *maxf*, we have $maxf(\mathcal{T}_t^\circ) = \max_{i=1}^{|E_t^\circ|} f(A_i, A_v \setminus A_i) = \max_{e \in E_t^\circ} \ell_v(e)$. Therefore, $F_{maxf}(maxf_{A_u}(t), \mathcal{T}_t, a) = F_{maxf}(\ell_u(t), \mathcal{T}_t, a)$ and $F_{maxf}(\ell_u(t), \mathcal{T}_t, a) = \max\{\ell_u(t), \max_{e \in E_t} \ell_v(e)\}$. Since the degree of t in T_v is bounded by k', F_{maxf} can be computed in constant time.

To show how to compute H_{maxf}, consider the step "merging the paths" of procedure *JoinNode* when applied to characteristics $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, dist_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, dist_w, out_w, branch_w, father_w) \in \mathrm{FSC}_{\mathrm{k,q}}(w)$ during the computation of a join node $v \in D$ with children u and w. Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be the partitioning tree for A_v obtained from the "merging" of $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$, a partitioning tree for A_u with $C_u = \mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$, and $(T_w^\circ, r_w^\circ, \sigma_w^\circ)$ with $C_w = \mathrm{Char}((T_w^\circ, r_w^\circ, \sigma_w^\circ), X_w)$, a partitioning tree for A_w.

For each edge e of T_v let $\mathcal{T}_e$ ($\mathcal{T}_e^\circ$) be the partition of X_v (A_v) that it defines. Similarly, for each vertex t of T_v let $\mathcal{T}_t$ ($\mathcal{T}_t^\circ$) be the partition of X_v (A_v) that it defines.

Assume by induction that for any edge e of T_u (T_w) we have $\ell_u(e) = maxf(\mathcal{T}_e^\circ \cap A_u)$ ($\ell_w(e) = maxf(\mathcal{T}_e^\circ \cap A_w)$) and that for any vertex t of T_u (T_w) we have $\ell_u(t) = maxf(\mathcal{T}_t^\circ \cap A_u)$ ($\ell_w(t) = maxf(\mathcal{T}_t^\circ \cap A_w)$).

Let $P_u(x, y)$ be a path of C_u that is merged with a path $P_w(x, y)$ of C_w obtaining $P_v(x, y)$. Let $P'_u(x, y)$ and $P'_w(x, y)$ be the extensions used in the merging. We set $\ell_v(e) = H_f(\ell_u(e), \ell_v(e), \mathcal{T}_e)$ to any edge of $P_v(x, y)$. From the fact that e defines a bipartition of A_v, we have $\ell_v(e) = maxf(\mathcal{T}_e^\circ)$.

For each internal vertex t of $P_v(x, y)$, let E_{t_u} (E_{t_w}) be the set of edges incident to t in T_u (T_w) and let $E_{t_u}^\circ$ ($E_{t_w}^\circ$) be the set of edges incident to t in T_u° (T_w°).

Then, for each vertex t in $P_v(x, y)$ we set $\ell_v(t) = H_{maxf}(\ell_u(t), \ell_w(t), \mathcal{T}_t) = \max\{\ell_u(t), \ell_w(t), \max_{e \in E_t} \ell_v(e)\}$. From the induction hypothesis, $\max\{\ell_u(t), \ell_w(t), \max_{e \in E_t} \ell_v(e)\} = \max\{\max_{e \in E_{t_u}^\circ} maxf(\mathcal{T}_e \cap A_u), \max_{e \in E_{t_w}^\circ} maxf(\mathcal{T}_e \cap A_w), \max_{e \in E_t} \ell_v(e)\} = maxf(\mathcal{T}_t^\circ)$.

Since the degree of t in T_v is bounded by k', H_{maxf} can be computed in constant time. □

Lastly, we need to show how to take into account the "structural" properties of different width. Hence, it is necessary to guarantee that characteristics belonging to $\mathrm{FSC}_{\mathrm{k,q}}(v)$ of a node v in the nice decomposition correspond to partitioning trees that satisfy these structural properties. For example, partitioning trees for the branch width are such that every internal vertex has degree three, therefore for each node v of the nice decomposition $\mathrm{FSC}_{\mathrm{k,q}}(v)$ must contain only characteristics of partitioning trees for A_v which have every internal vertex with degree three. We show how to modify the algorithm to take into account the structural properties for each width mentioned in Section 2.1. The procedure *StartingNode* is modified to compute only the characteristics that respects the "structural" properties of the width being computed. We show which changes to *IntroduceNode* procedure and *JoinNode* procedure are necessary in order to achieve this. Let $G = (V, E)$ be a graph and $(D, \mathcal{X})$ a nice decomposition of $A = E$ given to the algorithm as input, in the case of carving width and cut width, $(D, \mathcal{X})$ is a nice decomposition of $A = V$.

Tree width and Path width: no changes are made to the procedures.

Special tree width: in the *IntroduceNode* at after step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$", let $a = (x, y)$ be the element of $A_v \setminus A_u$ mapped by v_{leaf} and let X (Y) be the set of all leaves of T_v such that the edges of A_v they map have x (y) as one of its endpoints. Then, if the minimum spanning tree of T_v containing all vertices in X is not a caterpillar then the algorithm does not put C_v into $\mathrm{FSC}_{\mathrm{k,q}}(v)$ or if the minimum spanning tree of T_v containing all vertices in Y is not a caterpillar then the algorithm does not put C_v into $\mathrm{FSC}_{\mathrm{k,q}}(v)$. Procedure *JoinNode* remains unchanged.

Branch width, Linear width, Carving width and Cut width: we do not allow the Case 1 in the step "update of T_u into T_v" in the *IntroduceNode* procedure. That is, we do not allow a leaf mapping a to be added as neighbor of an internal vertex of T_u. In the *JoinNode* procedure during the step "merging the paths", we do not allow internal vertices of P_v, the result of merging P_u with P_w, to have a pair of vertices as its originators. In other words, any vertex of P_u must be "merged" with an edge of P_w and any vertex of P_w must be merged with an edge of P_u.

6.6 Time Complexity

The complexity of the algorithm to decide if a set A has q-branched Φ-width not bigger than k is given by the amount of time needed to compute a full set of characteristics for each node in the nice decomposition times the number of nodes in the nice decomposition. Let $(D, \mathcal{X})$ be the nice decomposition of A given as input to the algorithm. From the definition of a nice decomposition, we have that $|V(D)| = O(|A|)$. Let k' be equal to $\max_{X \in \mathcal{X}} |X|$. Hence, from Theorems 27, 29 and 31 the algorithm has time complexity bounded by

$$\max\{f_i(k, q, k'), f_f(k, q, k'), f_j(k, q, k')\} \cdot O(|A|).$$

Since the functions $f_i(k, q, k')$, $f_f(k, q, k')$ and $f_j(k, q, k')$ do not depend on $|A|$, if k, q and k' are given constants the algorithm has complexity bounded by $O(|A|)$. Therefore, it is a linear time algorithm when k, q and k' are fixed and it is a linear FPT-algorithm where the parameters are k, q and k'.

7 Conclusion

In this paper, we use a generalization of width parameters of graphs, the partition functions and partitioning trees, to design a unified FPT algorithm to decide if the q-branched tree width, special tree width, branch width, linear width, cut width and carving width of graphs are not bigger than an integer k.

Unfortunately, the algorithm presented only solves the decision problem. That is, it can be used to decide if the q-branched tree width, special tree width, branch width, linear width and cut width of a graph is not bigger than an integer k, but it does not compute the respective decomposition.

In [BK96], Bodlaender and Kloks propose an algorithm that decides if the tree width of a graph is at most a given integer k and, if it is the case, it constructs a tree decomposition with this width. This algorithm, which in part inspired our algorithm, also makes use of the notion of "characteristic" of a tree decomposition and proceeds to compute these "characteristics" by a dynamic

programming approach from a given tree decomposition of the graph. They also propose a second algorithm that constructs the tree decomposition from "characteristics" computed through the first algorithm.

In a current work, our algorithm was made constructive by following the same techniques used in their second algorithm, the one which constructs the tree decomposition for the input graph.

The algorithm we proposed in this paper can compute several graph width measures, while not being restricted to only the aforementioned width measures. For example, since the rank width of graphs can be defined in terms of partition functions and partitioning trees [AMNT09], we wonder whether using this formalization, is it possible to design a FPT-algorithm computing the rankwidth.

References

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[AMNT09] O. Amini, F. Mazoit, N. Nisse, and S. Thomassé. Submodular partition functions. *Discrete Mathematics*, 309(20):6000 – 6008, 2009.

[Bie91] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

[BK96] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[BK07] H. L. Bodlaender and A. M. C. A. Koster. On the maximum cardinality search lower bound for treewidth. *Discrete Applied Mathematics*, 155(11):1348–1372, 2007.

[BKK95] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. Discrete Math.*, 8(4):606–616, 1995.

[BL83] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM.

[BM93] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2):181–188, 1993.

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[BT97] H. L. Bodlaender and D. M. Thilikos. Constructive linear time algorithms for branchwidth. In *35th Int. Coll. on Automata, Languages and Programming*, volume 1256 of *LNCS*, pages 627–637, 1997.

[BT04] H. L. Bodlaender and D. M. Thilikos. Computing small search numbers in linear time. In *IWPEC*, volume 3162 of *LNCS*, pages 37–48, 2004.

[Cay89] A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.

[CM93] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.

[Cou10] B. Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS*, pages 13–29, 2010.

[FFN09] F. V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 53(3):358–373, 2009.

[FHL08] U. Feige, M. T. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

[FT03] F. V. Fomin and D. M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323–335, 2003.

[FT08] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[MN08] F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. *Theoretical Computer Science*, 399(3):169–178, 2008.

[Ree92] B. A. Reed. Finding approximate separators and computing tree width quickly. In *the 24th ACM Symposium on Theory of Computing (STOC)*, pages 221–228. ACM, 1992.

[RS83] N. Robertson and P. D. Seymour. Graph minors. i. Excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[RS86] Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.

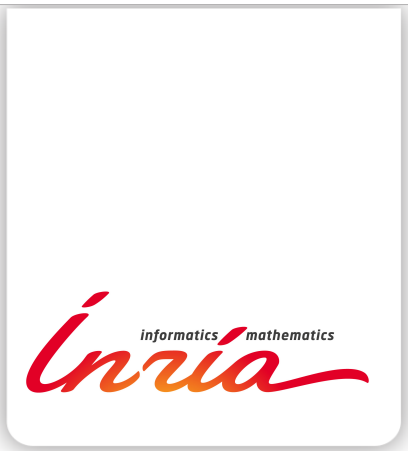
[RS91] N. Robertson and P. D. Seymour. Graph minors. x. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[RS04] N. Robertson and P. D. Seymour. Graph minors. xx. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[ST93] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.

[ST94] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[TSB00] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In *ISAAC*, volume 1969 of *LNCS*, pages 192–203, 2000.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399